НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

«На правах рукопису»
УДК 004.932

«До захисту допущено»
Науковий керівник кафедри
_____ І. А. Дичка

"___" _____ 2018 р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 121 "Інженерія програмного забезпечення"
на тему: «Моделі та програмне забезпечення для аналізування структури веб-застосунків на основі React.js»

Виконав:
студент VI курсу, групи КП-71мп
Рябченко Дмитро Олександрович _____
(підпис)

Науковий керівник:
Доцент кафедри програмного забезпечення
комп'ютерних систем ФПМ КПІ ім. Ігоря Сікорського,
к.т.н. Цуркан В.В. _____
(підпис)

Рецензент:
Начальник управління супроводження програми
інформатизації та нормативно-методичної діяльності
Департаменту інформатизації МВС України
к.т.н., доцент Дорогий Я.Ю. _____
(підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2018

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою

Спеціальність (спеціалізація) – 121 «Інженерія програмного забезпечення» («Програмне забезпечення комп'ютерних та інформаційно-пошукових систем»)

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ І.А. Дичка

«___»_____2018 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Рябченку Дмитру Олександровичу

1. Тема дисертації «Моделі та програмне забезпечення для аналізування структури веб-застосунків на основі React.js», науковий керівник дисертації Цуркан Василь Васильович, к.т.н., доцент, затверджені наказом по університету від «___»_____ 2018 р. №_____

2. Термін подання студентом дисертації «18» грудня 2018 р.

3. Об'єкт дослідження: процес аналізування структури веб-застосунків.

4. Предмет дослідження: програмне забезпечення для аналізування структури веб-застосунків на основі React.js.

5. Перелік завдань, які потрібно розробити:
   - Проаналізувати процес аналізування структури веб-застосунків.
   - Побудувати концептуальну модель програмного забезпечення аналізування структури веб-застосунків.
   - Побудувати структуру програмного забезпечення аналізування структури веб-застосунків.
   - Створити робочий проект програмного забезпечення для аналізування структури веб-застосунків на основі React.js.
   - Створити стартап проект програмного забезпечення для аналізування структури веб-застосунків на основі React.js.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу:
   - діаграма варіантів використання програмного забезпечення;
   - діаграма діяльності програмного забезпечення;
   - архітектура програмного забезпечення;
   - діаграма компонентів програмного забезпечення.

7. Орієнтовний перелік публікацій:
   - програмний засіб аналізування структури веб-застосунків на основі REACT.JS, XI наукова конференція магістрантів та аспірантів «Прикладна математика та комп'ютинг» (ПМК-2018-2).

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9. Дата видачі завдання «04» жовтня 2017 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Ґрунтовне ознайомлення з предметною галуззю	17.10.2017	
2.	Визначення структури магістерської дисертації; вивчення літератури	04.12.2017	
3.	Концептуальне моделювання програмного забезпечення аналізування структури веб-застосунків	15.02.2018	
4.	Створення структури програмного забезпечення аналізування структури веб-застосунків.	05.04.2018	
5.	Створення робочого проекту програмного забезпечення для аналізування структури веб-застосунків на основі React.js.	15.05.2018	
6.	Створення стартап проекту програмного забезпечення для аналізування структури веб-застосунків на основі React.js.	15.06.2018	
7.	Завершення роботи над магістерською дисертацією; підготовка ілюстративного матеріалу; підготовка матеріалів доповіді на конференції ПМК-2018	05.11.2018	
8.	Оформлення текстової і графічної частини магістерської дисертації	04.12.2018	

Студент Д.О. Рябченко

Науковий керівник дисертації В.В. Цуркан

РЕФЕРАТ

Актуальність теми. Над розробкою одного веб-застосунку може працювати велика команда розробників, а сам веб-застосунок може складатися з декількох тисяч компонентів React.js. Оскільки кожний розробник – людина, а людина та її працездатність залежить від багатьох факторів її повсякденного життя, то якість коду розробників може відрізнятися у різний проміжок часу. А якщо команда не вкладається у визначені часові рамки для розробки, то якість коду стає ще гіршою. Тому у таких випадках при створенні декількох компонентів для виконання одного функціоналу можливі надлишкове розширення структури об’єктів для відображення певної частини веб-застосунку в місцях, що насправді не потребують цього, або випадки нехтування корпоративними правилами розробки та недотримання цих правил. Таким чином, моделювання програмного забезпечення для аналізування структури веб-застосунків на основі React.js є актуальним.

Об’єктом дослідження є процес аналізування структури веб-застосунку на основі React.js.

Предметом дослідження є програмне забезпечення для аналізування структури веб-застосунків на основі React.js.

Мета роботи синтезувати програмне забезпечення для аналізування веб-застосунків на основі React.js.

Методи дослідження. Теоретичною основою дисертаційних досліджень є теорія моделювання процесів. Зокрема, теорія функціонального, процесного моделювання і моделювання потоків даних – для побудови концептуальної моделі програмного забезпечення аналізування структури веб-застосунків на основі React.js; теорія об’єктно-орієнтованого

моделювання – для побудови фізичної моделі програмного забезпечення для аналізування структури веб-застосунків на основі React.js.

Наукова новизна роботи полягає в одержанні таких результатів:

– уперше побудовано концептуальну модель програмного забезпечення аналізування структури веб-застосунків на основі React.js, використання якої дозволяє формалізувати його роботу на рівні функцій, процесів і потоків даних, а також сформувати та обґрунтувати варіанти використання програмного забезпечення;

– уперше побудовано фізичну модель програмного забезпечення на основі формалізування його роботи на рівні функцій, процесів і потоків даних, використання якої дозволяє надати нову якість програмному забезпеченню при створенні або вдосконаленні, зокрема, забезпечити його функціональну придатність до аналізування структури веб-застосунків на основі React.js.

Практична цінність отриманих результатів полягає у доведенні їх до практичного реалізування, а саме:

– розроблення програмного забезпечення аналізування структури веб-застосунків на основі React.js за його структурою;

– аналізування структури веб-застосунків на основі React.js завдяки використанню розробленого програмного забезпечення.

Апробація роботи. Результати роботи пройшли апробацію на XI науковій конференції магістрантів та аспірантів «Прикладна математика та комп’ютинг (ПМК-2018-2).

Структура та обсяг роботи. Магістерська дисертація складається зі вступу, п’яти розділів, висновків та додатків.

У вступі надано загальну характеристику роботи, виконано оцінку сучасного стану проблеми, обґрунтовано актуальність напрямку досліджень, сформульовано мету і задачі дослідження.

У першому розділі проаналізовано програмне забезпечення аналізування структури веб-застосунків. За результатами такого аналізування показано необхідність створення відповідного програмного забезпечення, сформовано функціональні та не функціональні вимоги для його синтезування.

У другому розділі побудовано концептуальну модель програмного забезпечення аналізування структури веб-застосунків на основі React.js на функціональному, процесному рівнях, рівні потоків даних у графічних нотаціях IDEF0, IDEF3 та DFD. Це дозволило на функціональному рівні формалізувати його роботу набором функцій. Тоді як на процесному рівні та рівні потоків даних описати взаємозв’язки між етапами аналізування структури веб-застосунків на основі React.js.

У третьому розділі на основі концептуальної моделі побудовано структуру програмного забезпечення аналізування структури веб-застосунків на основі React.js у графічній нотації UML. Це дозволило визначити його варіанти використання, логічну та фізичну структури та, як наслідок, синтезувати програмне забезпечення аналізування структури веб-застосунків на основі React.js.

У четвертому розділі визначено характеристики, метрики якості та проведено функціональне тестування програмного забезпечення аналізування структури веб-застосунків на основі React.js. Наведено контрольні приклади тестування основних його варіантів використання. Описано кроки успішного встановлення і наведено інструкцію щодо користування програмним забезпеченням аналізування структури веб-застосунків на основі React.js.

У п’ятому розділі визначено ринкові перспективи стартап-проекту програмного забезпечення аналізування структури веб-застосунків на основі React.js, графік та принципи організації виробництва, фінансовий аналіз та аналіз ризиків і заходи з просування пропозиції для інвесторів. Узагальнено етапи розроблення та виведення стартап-проекту на ринок.

У висновках проаналізовано отримані результати роботи.

У додатках наведено структуру програмного забезпечення та результати роботи програмного забезпечення аналізування структури веб-застосунків на основі React.js.

Магістерська дисертація виконана на 104 аркушах, містить 2 додатки та посилання на список використаних літературних джерел зі 44 найменувань. У роботі наведено 11 рисунків та 23 таблиці.

Ключові слова: веб-застосунок, структура веб-застосунку, аналізування структури веб-застосунку, React.js, DOM структура.

ABSTRACT

Actuality of theme. A large team of developers can work on the development of one web application, and the web application itself can consist of several thousand React.js components. Since each developer is a person, and the person and his ability to work depends on many factors in his everyday life, the quality of the developer code may vary in different periods of time. And, if the team does not fit into a defined time frame for development, then the quality of the code becomes even worse. Therefore, in such cases, when creating multiple components for a single functionality, there may be an excessive expansion of the object structure to display a certain part of the web application in places that do not really need it, or cases of corporate design rules ignoring and non-compliance with these rules. Thus, modeling software to analyze the structure of web applications based on React.js is relevant.

The object of the study is the process of analyzing the structure of a web application based on React.js.

The subject of the study is the software for analyzing the structure of web applications based on React.js.

The purpose of the work is to synthesize software for analyzing web applications based on React.js.

Research methods. The theoretical basis of dissertation research is the theory of process modeling. In particular, the theory of functional, process modeling and data flow modeling - for building a conceptual model of software for analyzing the structure of web applications based on React.js; the theory of object-oriented modeling - for building a physical software model for analyzing the structure of web applications based on React.js.

The scientific novelty of the work is to obtain the following results:

– for the first time a conceptual model of the software for analyzing the web application structure based on React.js is built, the use of which allows to formalize its work at the level of functions, processes and data flows, as well as to formulate and substantiate variants of software use;

– for the first time, the physical model of software was built on the basis of formalizing its work at the level of functions, processes and data flows, the use of which allows to provide new quality software when creating or improving, in particular, to ensure its functional suitability for analyzing the structure of web applications based on React.js.

The practical value of the results obtained is to bring them to practical realization, namely:

– developing software for analyzing the structure of web applications based on React.js according to its structure;

– analyzing the structure of web applications based on React.js through the use of the developed software.

Test work. The results of the work were tested at the 11th Conference of Masters and Postgraduate Students "Applied Mathematics and Computing" (PMK-2018-2).

Structure and scope of work. The master's thesis consists of an introduction, five sections, conclusions and appendices.

The introduction provides a general description of the work, evaluated the current state of the problem, substantiated the relevance of the research direction, formulated the purpose and objectives of the study.

The first section analyzes the software for analyzing the structure of web applications. According to the results of this analysis, the necessity of creating the

corresponding software has been demonstrated, functional and non-functional requirements for its synthesis have been formed.

In the second section, a conceptual model of the software for analyzing the structure of Web applications based on React.js on the functional, process levels, data streams in graphical notifications IDEF0, IDEF3 and DFD is constructed. This allowed the functional level to formalize its work by a set of functions. Then, at the process level and the data stream levels, describe the interrelationships between the stages of the analysis of the Web application structure on the basis of React.js.

In the third section, based on the conceptual model, the structure of the software for analyzing the web application structure based on React.js in the graphical UML notation is constructed. This made it possible to determine its usage patterns, logical and physical structure, and, as a result, to synthesize the software for analyzing the web application architecture based on React.js.

The fourth section defines the characteristics, quality metrics, and performs functional testing of the software for analyzing the web application architecture based on React.js. Examples of testing their main uses are given. Describes the steps for successful installation and provides instructions on how to use the React.js based web application structure analysis software.

The fifth section defines the market prospects of the startup project's software to analyze the structure of web applications based on React.js, the schedule and principles of production organization, financial analysis and risk analysis and measures to promote the proposal to the investor.

The conclusions are analyzed the results of work.

The annexes present the software architecture and the results of the software for analyzing the web application structure based on React.js.

The master's thesis is made on 104 sheets, contains 2 appendices and a link to the list of used literary sources from 44 titles. There are 11 drawings and 23 tables in the work.

Keywords: Web Application, Web Application Structure, Web Application Structure Analysis, React.js, DOM Structure.

РЕФЕРАТ

Актуальность темы. Над разработкой одного веб-приложения может работать большая команда разработчиков, а само веб-приложение может состоять из нескольких тысяч компонентов React.js. Поскольку каждый разработчик - человек, а человек и его работоспособность зависит от многих факторов его повседневной жизни, то качество кода разработчиков может отличаться в разные промежутки времени. А если команда не укладывается в определенные временные рамки для разработки, то качество кода становится еще хуже. Поэтому в таких случаях при создании нескольких компонентов для выполнения одного функционала возможные избыточное расширение структуры объектов для отображения определенной части веб-приложения в местах, которые на самом деле не нуждаются в этом, или случаи пренебрежения корпоративными правилами разработки и несоблюдение этих правил. Таким образом, моделирование программного обеспечения для анализа структуры веб-приложений на основе React.js является актуальным.

Объектом исследования является процесс анализа структуры веб-приложения на основе React.js.

Предметом исследования является программное обеспечение для анализа структуры веб-приложений на основе React.js.

Цель работы синтезировать программное обеспечение для анализа веб-приложений на основе React.js.

Методы исследования. Теоретической основой диссертационных исследований является теория моделирования процессов. В частности, теория функционального, процессного моделирования и моделирования потоков данных - для построения концептуальной модели программного обеспечения анализа структуры веб-приложений на основе React.js; теория объектно-

ориентированного моделирования - для построения физической модели программного обеспечения для анализа структуры веб-приложений на основе React.js.

Научная новизна работы заключается в получении следующих результатов:

– впервые построено концептуальную модель программного обеспечения анализа структуры веб-приложений на основе React.js, использование которой позволяет формализовать его работу на уровне функций, процессов и потоков данных, а также сформировать и обосновать варианты использования программного обеспечения;

– впервые построены физическую модель программного обеспечения на основе формализации его работы на уровне функций, процессов и потоков данных, использование которой позволяет предоставить новое качество программному обеспечению при создании или совершенствовании, в частности, обеспечить его функциональную пригодность к анализу структуры веб-приложений на основе React. js.

Практическая ценность полученных результатов заключается в доведении их до практической реализации, а именно:

– разработка программного обеспечения анализа структуры веб-приложений на основе React.js по его структуре;

– анализ структуры веб-приложений на основе React.js благодаря использованию разработанного программного обеспечения.

Апробация работы. Результаты работы прошли апробацию на XI научной конференции магистрантов и аспирантов «Прикладная математика и компьютинг» (ПМК-2018-2).

Структура и объем работы. Магистерская диссертация состоит из введения, пяти глав, заключения и приложений.

Во введении дана общая характеристика работы, выполнена оценка современного состояния проблемы, обоснована актуальность направления исследований, сформулированы цели и задачи исследования.

В первой главе проанализированы программное обеспечение анализа структуры веб-приложений. По результатам такого анализа показана необходимость создания соответствующего программного обеспечения, сформирован функциональные и нефункциональные требования для его синтезирования.

Во втором разделе построено концептуальную модель программного обеспечения анализа структуры веб-приложений на основе React.js на функциональном, процессном уровне, уровне потоков данных в графической нотации IDEF0, IDEF3 и DFD. Это позволило на функциональном уровне формализовать его работу набором функций. Тогда как на процессном уровне и уровне потоков данных описать взаимосвязи между этапами анализа структуры веб-приложений на основе React.js.

В третьем разделе на основе концептуальной модели построены структуру программного обеспечения анализа структуры веб-приложений на основе React.js в графической нотации UML. Это позволило определить его варианты использования, логическую и физическую структуры и, как следствие, синтезировать программное обеспечение анализа структуры веб-приложений на основе React.js.

В четвертом разделе определены характеристики, метрики качества и проведено функциональное тестирование программного обеспечения анализа структуры веб-приложений на основе React.js. Приведены контрольные примеры тестирование основных его вариантов использования. Описаны шаги успешной установки и приведена инструкция по пользованию

программным обеспечением анализа структуры веб-приложений на основе React.js.

В пятом разделе определены рыночные перспективы стартап-проекта программного обеспечения анализа структуры веб-приложений на основе React.js, график и принципы организации производства, финансовый анализ и анализ рисков и меры по продвижению предложения для инвесторов. Обзор этапа разработки и вывода стартап-проекта на рынок.

В выводах проанализированы полученные результаты работы.

В приложениях приведена структура программного обеспечения и результаты работы программного обеспечения анализа структуры веб-приложений на основе React.js.

Магистерская диссертация выполнена на 104 листах, содержит 2 приложения и ссылки на список использованных литературных источников из 44 наименований. В работе приведены 11 рисунков и 23 таблицы.

Ключевые слова: веб-приложение, структура веб-приложения, анализ структуры веб-приложения, React.js, DOM структура.

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

Веб-застосунок – розподілений застосунок, в якому клієнтом виступає браузер, а сервером — веб-сервер. Браузер може бути реалізацією так званих тонких клієнтів — логіка застосунку зосереджується на сервері, а функція браузера полягає переважно у відображенні інформації, завантаженої мережею з сервера, і передачі назад даних користувача. Однією з переваг такого підходу є той факт, що клієнти не залежать від конкретної операційної системи користувача, тому веб-застосунки є міжплатформовими сервісами[1].
React.js – відкрита JavaScript бібліотека для створення інтерфейсів користувача, яка покликана вирішувати проблеми часткового оновлення вмісту веб-сторінки, з якими стикаються в розробці односторінкових застосунків. Розробляється Facebook, Instagram і спільнотою індивідуальних розробників. React дозволяє розробникам створювати великі веб-застосунки, які використовують дані, котрі змінюються з часом, без перезавантаження сторінки. Його мета полягає в тому, щоб бути швидким, простим, масштабованим. React обробляє тільки користувацький інтерфейс у застосунках. Це відповідає видові у шаблоні модель-вид-контролер (MVC), і може бути використане у поєднанні з іншими JavaScript бібліотеками або в великих фреймворках MVC, таких як AngularJS. Він також може бути використаний з React на основі надбудов, щоб піклуватися про частини без користувацького інтерфейсу побудови веб-застосунків [2].

ВСТУП

Розробка програмного забезпечення стала масштабних розмірів. Існують проекти над якими працює велика команда розробників, дизайнерів, аналітиків. Кожна частина команди відповідальна за кінцевий результат, програмний засіб чи систему, що розробляються.

Для розробки програмного забезпечення створено багато методик, метою яких є прискорення робочого процесу та налагодження співпраці між всіма розробниками, що працюють над спільним проектом. Більш того, великі компанії-виробники програмного забезпечення складають власні зводи правил, яких необхідно притримуватись при написані нового коду в рамках проекту. Ці правила здебільшого стосуються найменуванню змінних, функцій, класів. Тоді як стилістичні правила орієнтовані на підтримання читабельності коду іншими розробниками.

Але всі ці правила та методики не здатні запобігти наявності людського фактору, адже над кожним програмним забезпеченням працюють люди. Завжди можливі ситуації, або навіть сукупність ситуацій, кожна з яких впливає на працездатність людини. Тому будь-яка методика розробки програмного забезпечення чи звід корпоративних правил не можуть гарантувати їх повноцінного використання кожним членом команди.

Таким чином, моделювання програмного забезпечення для аналізування структури веб-застосунків на основі React.js є актуальним.

1 ПРОБЛЕМАТИКА ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АНАЛІЗУВАННЯ СТРУКТУРИ ВЕБ-ЗАСТОСУНКІВ НА ОСНОВІ REACT.JS

1.1. Розгляд процесу аналізування структури веб-застосунків на основі React.js

Надійність програмного забезпечення - це підвищений ризик загальної надійності системи. Оскільки системи розширюються та стають більш складними, функціональність у системах, що критично важлива для безпеки, частіше всього контролюється програмним забезпеченням. Ця зміна разом із підвищенням надійності в апаратних модулях змістила основну причину відмови системи від обладнання до програмного забезпечення.

Тому аналізування програмного забезпечення, а саме аналізування вихідного коду, є однією з важливих частин процесу розробки програмного забезпечення.

Статичний аналіз - це техніка, яка може підвищити якість та надійність програмного забезпечення вбудованих систем. Інтеграція інструментів та методів статичного аналізу в процес розробки може призвести до суттєвого скорочення тестів на розробку та збоїв в роботі. Проте інтеграція статичного аналізу в процес розробки може бути складною, особливо, якщо в проектах використовується велика кількість застарілого коду [3-5].

Статичний аналіз вихідного коду - це широкий термін для набору методів, що використовуються для перевірки комп'ютерного програмного забезпечення без фактичного виконання програм. Витонченість аналізу сильно залежить від використовуваного інструмента. Найпростіші інструменти часто лише пошук вихідного коду для текстового шаблону відповідають або розраховують основні показники програми, щоб визначити ймовірність виникнення проблем з певного сегмента коду. Більш просунуті

інструменти статичного аналізу виконують роль вдосконаленого компілятора для вихідного коду, глибоко аналізуючи як виконання, так і потоки даних для несправностей, які можуть призвести до невдачі. Деякі з найсучасніших інструментів також включатимуть інформацію про посилання в їх аналізі, щоб визначити проблеми на вищому рівні.

Статичний аналіз коду є частиною того, що називається тестуванням білого ящику, так як, на відміну від тестування чорного ящику, вихідний код доступний для тестувальників. Багато видів тестування програм включають статичний аналіз коду, де розробники та інші сторони шукають помилки або іншим способом аналізують код програмного забезпечення. Різні види статичного аналізу коду включають тестування на різних рівнях, наприклад, на рівні одиниці або на рівні системи.

Статичний аналіз вихідного коду не представляє нової технології. Як правило, використовується під час виконання та огляду для виявлення помилок впровадження програмного забезпечення, було показано, що статичний аналіз зменшує дефекти програмного забезпечення в 6 разів, а також виявляє 60% відмов після випуску.

Залежно від використовуваного спеціального інструменту, методи статичного аналізу можуть визначати переповнення буферів, уразливості системи безпеки, витік пам'яті, аномалії часу (наприклад, умови перегонів, тупикові місця та ліфти), мертві або невикористані сегменти вихідного коду та інші загальні помилки в програмі. Недавні дослідження показали зв'язок між несправностями, виявленими під час автоматизованої перевірки, і фактичною кількістю несправностей на місцях, що виникають у конкретному продукті.

Експерти вказують, що крок компіляції, виконаний сучасними компіляторами, є формою статичного аналізу коду, оскільки він призначений

для виявлення різних типів синтаксичних чи технічних помилок перед запуском програми. Ресурси статичного аналізу коду повинні забезпечити кращу якість коду, хоча деякі фахівці з інформаційних технологій стверджують, що можуть існувати проблеми із таким типом тестування, деякі з яких пов'язані з надто стандартизованими засобами налагодження. Крім того, хоча такі інструменти, як компілятори, можуть виявляти багато видів синтаксичних помилок, статичне тестування коду може призвести до виникнення ширших логічних помилок, які можуть поставити під загрозу якість. Деякі з них повинні бути знайдені в динамічному тестуванні коду. [3]

Динамічний аналіз коду - це процедура тестування, яка є частиною процесу налагодження програмного забезпечення та використовується для оцінки програми під час виконання в реальному часі. Він застосовується на етапі розробки. Основна мета динамічного аналізу коду полягає в тому, щоб знайти помилки під час запуску програми, після виклику функції, де змінним були призначенні певні значення, порівняно з перевіркою кожної рядка коду, миттєвим застосуванням значень та уявленнями про можливі сценарії розгалуження. Як основна частина налагодження коду, динамічний аналіз коду дозволяє тестувати програму за будь-яким сценарієм, усуваючи необхідність штучно створювати входи або ситуації, які можуть призвести до непередбачених ефектів або помилок. Це стандартна процедура, оскільки вона знижує вартість та час тестування, а також сприяє технічному обслуговуванню. Цей процес може виявити несподівані проблеми, як-от певні непотрібні вимоги або необхідні функції. Такі проблеми не виявляються під час процесу проектування, тому що люди не можуть визначити всі можливі сценарії. Що може здатися добре на папері, це не завжди перекладається на практиці та під час спостережуваного виконання в реальному часі. Динамічний аналіз коду також застосовується на етапі тестування, коли

інженери виявляють певні помилки, які з'являються лише після декількох виконань або в окремих сценаріях. У таких випадках єдиним вибором є динамічне тестування програми та відтворення сценаріїв [4].

Розглянуті в даному розділі види аналізу вихідного коду не враховують структуру програмного забезпечення, тому можемо стверджувати про необхідність синтезу такого виду аналізу, об’єктом якого буде саме структура програмного забезпечення, шляхом розгляду існуючих рішень аналізування програмного забезпечення у п. 1.2 та визначенні додаткових вимог у п. 1.3.

1.2. Розгляд програмних засобів аналізування структури веб-застосунків на основі React.js

Проблема перевірки якості вихідного коду виникла відразу після створення мов програмування та можливості створювати програмне забезпечення за допомоги мов програмування. Але перші декілька десятиліть розроблені системи не були такими великими, як системи, що зараз існують або створюються. Тому тоді для аналізу якості вихідного коду було достатньо компілятора, який в свою чергу виконує лексичний, синтаксичний, семантичний аналізи.

З ростом обсягу програмного забезпечення виникають нові проблеми та нові вимоги до аналізування вихідного коду. Оскільки сучасні системи мають мільйони строк вихідного коду та над розробкою працюють сотні розробників, однією з важливих частин аналізування вихідного коду є стилістичний аналіз, який дозволяє зберігати читабельність, що в свою чергу прискорює знайомство нового розробника з системою та поліпшує процес підтримки існуючого програмного забезпечення. В рамках даної магістерської дисертації об’єктом аналізування є веб-застосунки, створенні за допомогою мови програмування Javascript з використанням бібліотеки

React.js для створення графічного інтерфейсу, тому в даному розділі будуть розглядатись найбільш популярні серед розробників існуючі рішення для аналізування вихідного коду, що підтримують мову програмування Javascript.

ESLint - це лінтувальна утиліта з відкритим вихідним кодом JavaScript, створена в червні 2013 р. Ніколасом С. Закасом. Лінтування коду - це тип статичного аналізу, який часто використовується для пошуку проблемних моделей або коду, який не відповідає певним правилам щодо стилю. Для більшості мов програмування існують лінтувальні утиліти, а деякі компілятори іноді включають їх в процес компіляції.

JavaScript, який є динамічною та вільно набраною мовою, особливо піддається помилці розробника. Без користі процесу компіляції код JavaScript, як правило, виконується для пошуку синтаксису або інших помилок. Інструменти з нанесенням, такі як ESLint, дозволяють розробникам виявляти проблеми з кодом JavaScript, не виконуючи його.

Основна причина створення ESLint полягає в тому, щоб дозволити розробникам створювати власні правила лінтування. ESLint призначений для того, щоб всі правила повністю підключалися. Правила за замовчуванням записуються так само, як і будь-які правила плагінів. Всі вони можуть виконувати одну і ту ж схему, як для самих правил, так і для тестів. Хоча ESLint буде поставляти з деякими вбудованими правилами, щоб зробити їх корисним з самого початку, ви зможете динамічно завантажувати правила в будь-який момент часу [6].

JSHint. Проект має на меті допомогти розробникам JavaScript створювати складні програми, не турбуючись про помилки та мови.

Будь-яка кодова база зрештою стає величезною в певний момент, тому прості помилки - які не відображаються під час написання – можуть зруйнувати презентацію системи та додавати ще декілька годин

налагодження. Таким чином, інструменти для статичного аналізу коду вступають в гру і допомагають розробникам виявляти такі проблеми. JSHint сканує програму, написану за допомогою мови програмування JavaScript, і повідомляє про загальні помилки та можливі помилки. Потенційною проблемою може бути синтаксична помилка, помилка, пов'язана з неявним перетворенням типу, змінною, що призводить до витоку даних або цілком іншим.

Лише 15% всіх програм, наведених на jshint.com, успішно проходять перевірку JSHint. У всіх інших випадках JSHint знаходить деякі червоні прапори, які могли б бути помилками або потенційними проблемами [7].

Розглянуті в даному розділі програмні засоби для аналізування веб-застосунків здатні виконувати лінтування вихідного коду, допомагаючи розробникам запобігти появлення деяких помилок ще до запуску веб-застосунку. Але жоден з них не розглядає відношення між компонентами веб-застосунку, що в свою чергу підтверджує актуальність даної магістерської дисертації та необхідність у створенні програмного засобу для аналізування структури веб-застосунків.

1.3. Визначення вимог до програмного засобу аналізування структури веб-застосунків на основі React.js

Веб-застосунки на основі React.js складаються з компонентів графічного інтерфейсу, які можна використовувати в різних місцях веб-застосунку. Для читабельності та підтримки компонентів кожен компонент описується в окремому файлі, а деколи і в окремих папках. Через це стає неможливим побудувати структуру веб-застосунку, що аналізується. Структуру веб-застосунку можна побудувати тільки під час запуску. Тому

програмний засіб, що є предметом даної магістерської дисертації не можна віднести повністю до статичного аналізу веб-застосунків на основі React.js.

Але з іншого боку, метою нашого програмного засобу є аналізування структури веб-застосунку на основі React.js, яке не потребує виконання функцій, не залежить від значень, що були призначені змінним, не виявляє витік даних чи проблеми у безпеці веб-застосунків. Тому і до динамічного аналізування наш програмний засіб також не можна віднести.

Кожний програмний засіб має власні особливості. Так і кожна команда розробників має власні корпоративні правила, які необхідно дотримуватись при розробці. Натхненні прикладом ESLint, для підтримки таких унікальностей було вирішено використовувати шаблони для аналізування. Таким чином кожна команда розробників буде мати можливість створювати власний набір шаблонів для свого програмного забезпечення [8].

В рамках магістерської дисертації буде розроблено програмний засіб для аналізування структури веб-застосунків на основі React.js, що складається з наступних модулів:

– модуль парсингу веб-застосунків на основі React.js;

– модуль аналізування веб-застосунків на основі React.js;

– модуль відображення структури веб-застосунку та результатів аналізування.

Основні функціональні вимоги до програмного засобу наведено в табл. 1.1.

Для визначення коректності вимог в цілому для системи необхідно за допомоги матриці залежності встановити чи не суперечать вони одна одній та чи не перекриваються між собою.

Матрицю залежності вимог до програмного засобу аналізування структури веб-застосунків на основі React.js наведено в табл. 1.2, яка вказує на те, що жодних протиріч або накладань вимог не виявлено.

Таблиця 1.1

Функціональні вимоги до програмного забезпечення для аналізування структури веб-застосунків на основі React.js

Кодовий номер	Вимога
B1	Можливість аналізувати структуру веб-застосунку на основі React.js
B2	Зчитування шаблонів для аналізу
B3	Парсинг структури веб-застосунків на основі React.js
B4	Відображення структури веб-застосунку на основі React.js
B5	Відображення результатів аналізу структури веб-застосунку на основі React.js
B6	Можливість перегляду вершини у структурі, де було виявлено проблему

Таблиця 1.2

Матриця залежності функціональних вимог до програмного забезпечення для аналізування структури веб-застосунків на основі React.js

	B1	B2	B3	B4	B5	B6
B1	X	X	X	X	X	X
B2		X	X	X	X	X
B3			X	X	X	X
B4				X	X	X
B5					X	X
B6						X

1.4. Висновки

У цьому розділі розглянуто процес аналізування програмного забезпечення, в результаті якого було виділено статичний та динамічний аналіз вихідного коду. Після цього розглянуто існуючі рішення статичного аналізу, опираючись на принцип та призначення їх роботи, що в свою чергу підтвердило актуальність даної магістерської дисертації та необхідність створення програмного засобу для аналізування структури веб-застосунків на основі React.js. Також було визначено функціональні вимоги до програмного засобу аналізування структури веб-застосунку на основі React.js та отриманий список вимог було перевірено на наявність протиріч чи накладань.

Основні результати розділу опубліковано в роботі [8].

2 КОНЦЕПТУАЛЬНА МОДЕЛЬ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АНАЛІЗУВАННЯ СТРУКТУРИ ВЕБ-ЗАСТОСУНКІВ НА ОСНОВІ REACT.JS

2.1. Формалізування процесу аналізування структури веб-застосунків на основі React.js

У попередньому розділі підтвердили актуальність обраної теми для магістерської дисертації та визначили функціональні вимоги до програмного засобу для аналізування структури веб-застосунків на основі React.js. Наступним кроком для реалізації програмного засобу є формалізація самого процесу аналізування структури веб-застосунків на основі React.js [8], метою якого є спростити розуміння предметної області та завдань, що ставляться перед програмним засобом, який буде розроблено в рамках магістерської дисертації. Для цього будемо використовувати методику функціонального моделювання та графічного опису процесів IDEF0.

IDEF0 – методологія функціонального моделювання і графічного описання процесів, призначена для формалізації і опису бізнес-процесів. Особливістю IDEF0 є її акцент на ієрархічне представлення об'єктів шляхом розбиття процесу на під процеси, що значно полегшує розуміння предметної області. В IDEF0 розглядаються логічні зв'язки між роботами, а не послідовність їх виконання в часі [9].

Процес аналізування структури веб-застосунків на основі React.js зображено за допомогою діаграми A-0 в нотації IDEF0 та її декомпозицій на окремі функції за допомогою діаграм A0 та A1 в нотації IDEF0. Дані діаграми зображені на рис. 2.1 та 2.2 [8].

Діаграма A-0 демонструє найбільш абстрактний опис програмного засобу для аналізування структури веб-застосунків на основі React.js, на якій вказаний сам суб’єкт моделювання, обмеження, які впливають на виконання

процесу, вхідні дані, які необхідні для виконання самого процесу, вихідні дані, тобто результат виконання процесу аналізування структури веб-застосунків на основі React.js та механізми – ресурси, що виконують роботу.

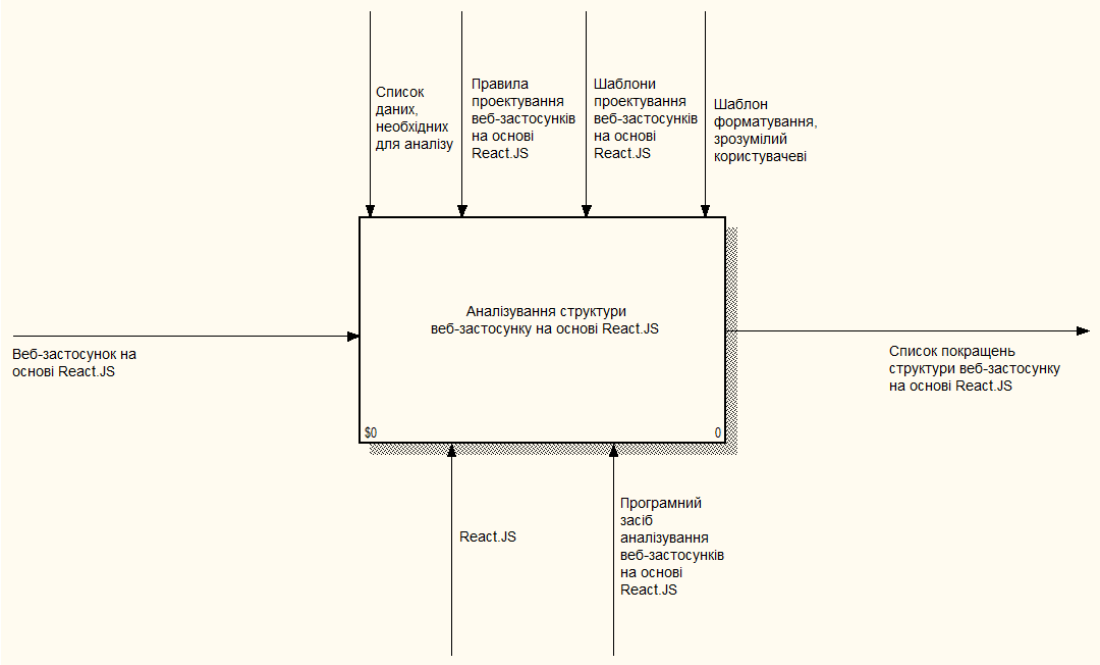


Рис. 2.1. Функціональна модель програмного забезпечення для аналізування структури веб-застосунків на основі React.js

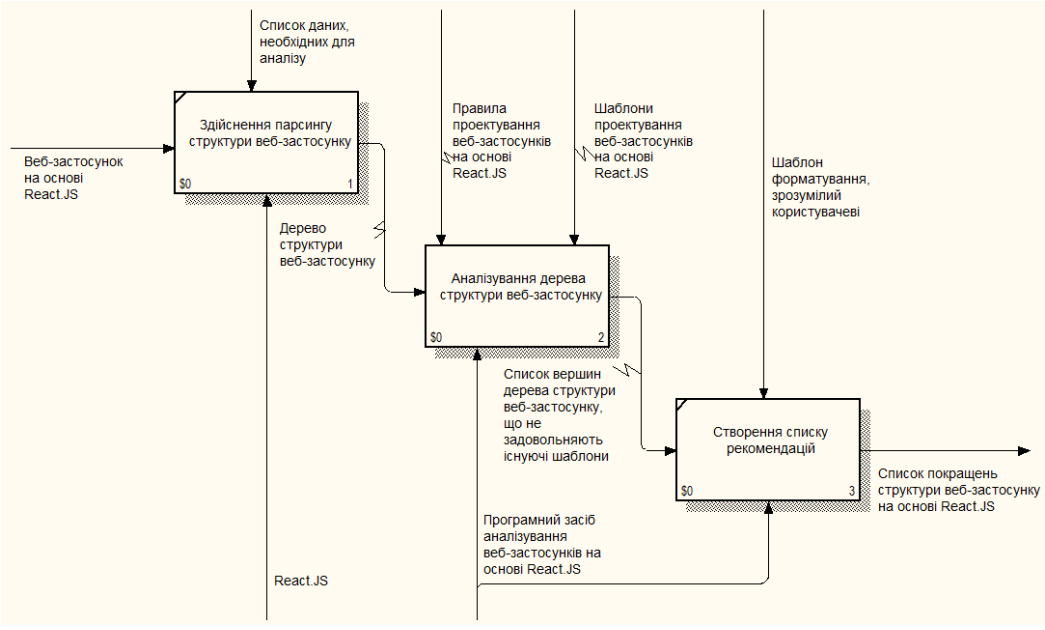


Рис. 2.2. Декомпозиція функціональної моделі програмного забезпечення

Згідно з функціональною моделлю аналізування структури веб-застосунків на основі React.js, зображеною на рис. 2.1., можна побачити, що вхідними даними виступає веб-застосунок на основі React.js та на виході ми отримуємо список покращень структури веб-застосунку на основі React.js.

Діаграма А-0 має наступні обмеження:

- список даних, необхідних для аналізу;
- правила проектування веб-застосунків на основі React.js;
- шаблони проектування веб-застосунків на основі React.js;
- шаблони форматування, зрозумілий користувачеві.

Механізмами, що виконують процес, є:

- React.js;
- програмний засіб аналізування веб-застосунків на основі React.js.

Діаграма А0 є декомпозицією контекстної діаграми, вона виділяє основні функції системи, вказує потоки даних між ними, вхідні та вихідні дані для різних функцій, які можуть керувати даними функціями.

Декомпозиція функціональної моделі, зображена на рис. 2.2., має основні функції такі як:

- здійснення парсингу структури веб-застосунку(А-1);
- аналізування дерева структури веб-застосунку(А-2);
- створення списку рекомендацій(А-3).

Блок А-1 «Здійснення парсингу структури веб-застосунку» діаграми А0 приймає веб-застосунок на основі React.js як вхідні дані. Даний процес виконується бібліотекою React.js, а саме допоміжною утилітою від тих самих розробників – react-devtools. На його роботу впливає список даних, необхідних для аналізу, тобто набір даних, що зберігається у кожному компоненті веб-застосунку на основі React.js і є необхідними для визначення відношень між вершинами структури. Результатом виконання є дерево

структури веб-застосунку, кожна вершина якого містить необхідний набір даних для визначення відношень між його вершинами та порівняння цих відношень з відповідними відношеннями, визначеними у шаблонах для аналізу.

Блок А-2 «Аналізування дерева структури веб-застосунку» приймає на вхід результат роботи попереднього процесу, а саме дерево структури веб-застосунку. Під впливом правил проектування веб-застосунків, тобто рекомендацій від розробників бібліотеки React.js та видатних розробників у сфері веб-застосунків та шаблонів проектування веб-застосунків на основі React.js, створених розробниками або керівництвом певної компанії та які відтворюють корпоративні правила цієї компанії, що буде використовувати розроблений в рамках магістерської дисертації програмний засіб для аналізування структури веб-застосунків на основі React.js, процес аналізування дерева структури веб-застосунку синтезує список вершин дерева структури веб-застосунку, що не задовольняють існуючі шаблони.

Блок А-3 «Створення списку рекомендацій» отримує список вершин дерева структури веб-застосунку виконує його форматування згідно із шаблоном форматування, створений для відображення повідомлення помилки та положення вершини у структурі веб-застосунку таким чином, щоб це було зрозуміло користувачеві.

Діаграма А1, представлена на рис. 2.3. [8], є декомпозицією блоку А2 з діаграми А0, показує дії необхідні для виконання процесу А2 – «Аналізування дерева структури веб-застосунку» та складається з наступних функцій:

– «Створення під дерева для аналізу» - проходячи через кожну вершину дерева структури веб-застосунку будується під дерево з початком у поточній вершині дерева структури веб-застосунку у

відповідності до шаблону чи правила, з яким буде виконано порівняння;

– «Створення під дерева на основі шаблону» - створюється на основі шаблонів, створених користувачами програмного засобу, та використовується для порівняння з під деревом, утвореного з дерева структури веб-застосунку, що аналізується;

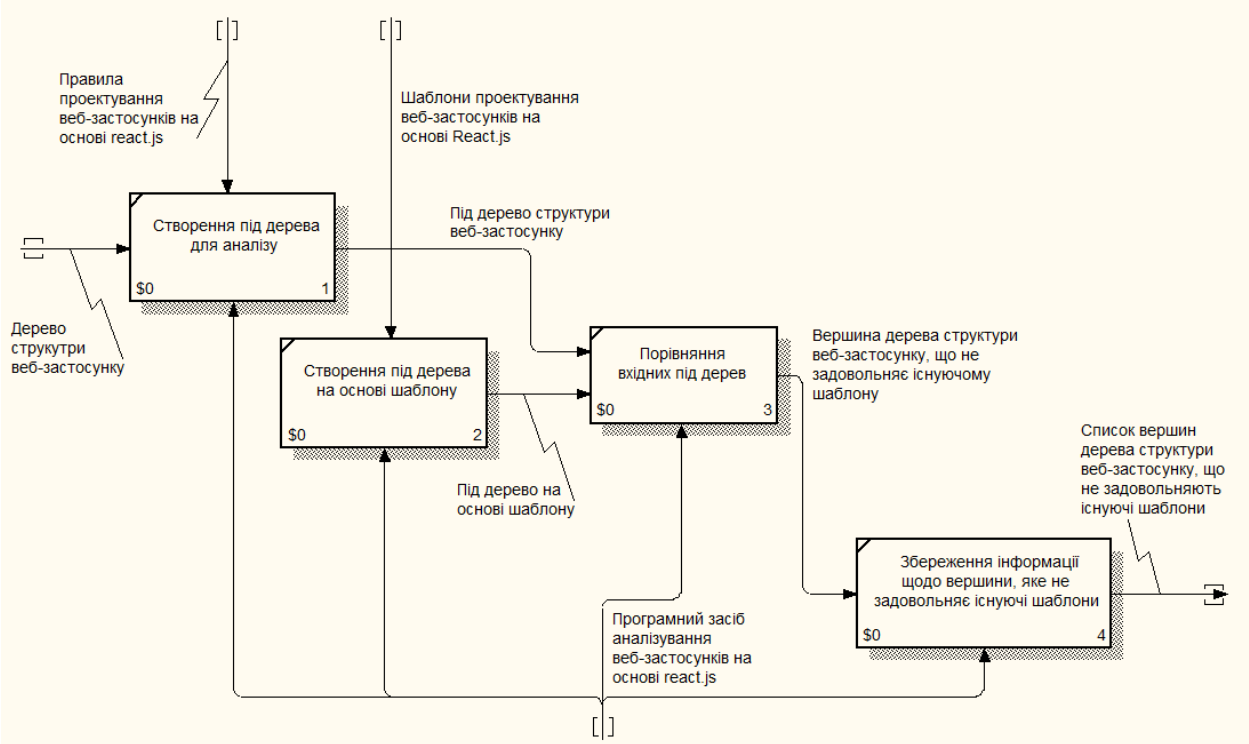


Рис. 2.3. Декомпозиція процесу А2 – «Аналізування структури веб-застосунку»

– «Порівняння вхідних під дерев» - створивши під дерево зі структури веб-застосунку, що аналізується, та з шаблону, з яким буде порівнюватись перше під дерево, виконується співставлення характеристик, котрі містяться у кожній вершині під дерева;

– «Збереження інформації щодо вершини, яке не задовольняє існуючі шаблони» - інформація про кожне під дерево, яке містить вершину дерева структури веб-застосунку, та яке не задовольняє хоча б одному існуючому шаблону, зберігається разом для

подальшого форматування на наступному кроці процесу аналізування структури веб-застосунків на основі React.js.

На основі найнижчого рівня декомпозиції можливо зробити висновок для формування необхідної архітектури програмного засобу та його конструювання.

2.2. Визначення зв’язків між етапами аналізування структури веб-застосунків на основі

Після того, як ми формалізували функціональну модель для процесу аналізування структури веб-застосунків на основі React.js, нам необхідно визначити зв’язки між етапами роботи нашого програмного засобу. Для цього було використано методику функціонального моделювання та графічного опису процесів IDEF3, яка представляє механізм документування та збору інформації про процес аналізування структури веб-застосунків на основі React.js.

IDEF3 – це метод моделювання бізнес-процесу, який доповнює IDEF0. Метод IDEF3 - це метод опису потоку технологічних процесів, керований сценарієм, призначений для захоплення знань про те, як працює певна система.

Одним із основних механізмів, що використовуються для опису світу – зв'язок історії з послідовністю подій або заходів. Метод засвоєння опису процесу IDEF3 був створений для засвоєння описів послідовностей дій, які вважаються загальними механізмами для опису ситуації чи процесу. Основною метою IDEF3 є надання структурованого методу, за допомогою якого можна висловити знання про роботу конкретної системи або організації. Засвоєння знань відбувається шляхом безпосереднього засвоєння тверджень про реальні процеси та події у формі, яка є найбільш природною

для засвоєння. IDEF3 підтримує цей вид придбання знань, забезпечуючи надійний та добре структурований підхід до придбання знань про процес та експресивну, але просту у використанні мову для засвоєння та виявлення інформації [10].

На рис. 2.4 можна побачити декомпозицію контекстної діаграми аналізування структури веб-застосунків на основі React.js (рис. 2.1) в нотації IDEF3, на якій зображені зв’язки між такими під процесами, як [8]:

– здійснення парсингу структури веб-застосунку;
– аналізування дерева структури веб-застосунку;
– створення списку рекомендацій.

Оскільки це перший рівень декомпозиції, все під процеси виконуються послідовно, тобто кожен під процес очікує завершення попереднього під процесу.

Основні дії з даними, а саме їх аналізування, виконується блоці «Аналізування дерева структури веб-застосунку» [8], тому розглянемо декомпозицію саме цього під процесу. На рис. 2.5 зображено зв’язки між етапами роботами процесів «Аналізування дерева структури веб-застосунку»:

– побудова дерева структури веб-застосунку;
– побудова під дерева для аналізу;
– побудова під дерева на основі шаблону;
– порівняння під дерев;
– збереження під дерева, якщо порівняння співпало.

Процес цієї декомпозиції починається з побудови дерева структури веб-застосунку. Після завершення побудови відбувається прохід через кожну вершину отриманого дерева та, зупиняючись на кожній вершині дерева, будується під дерево, яке порівнюється з існуючими шаблонами аналізування структури веб-застосунків. Шаблони в свою чергу також представляються як

під дерева. І тільки після створення обох під дерев відбувається їх порівняння. У випадку, якщо під дерево, що було створено на основі структури веб-застосунку, та під дерево, створене на основі шаблону для аналізу, співпадають, ми можемо стверджувати, що у цьому під дереві знайдено помилку, яка відповідає шаблону, з яким було виконано порівняння. Інформація про таку вершину зберігається та передається до наступного кроку процесу аналізування структури веб-застосунків на основі React.js, де всі збереженні вершини будуть відформатовані для відображення користувачеві.

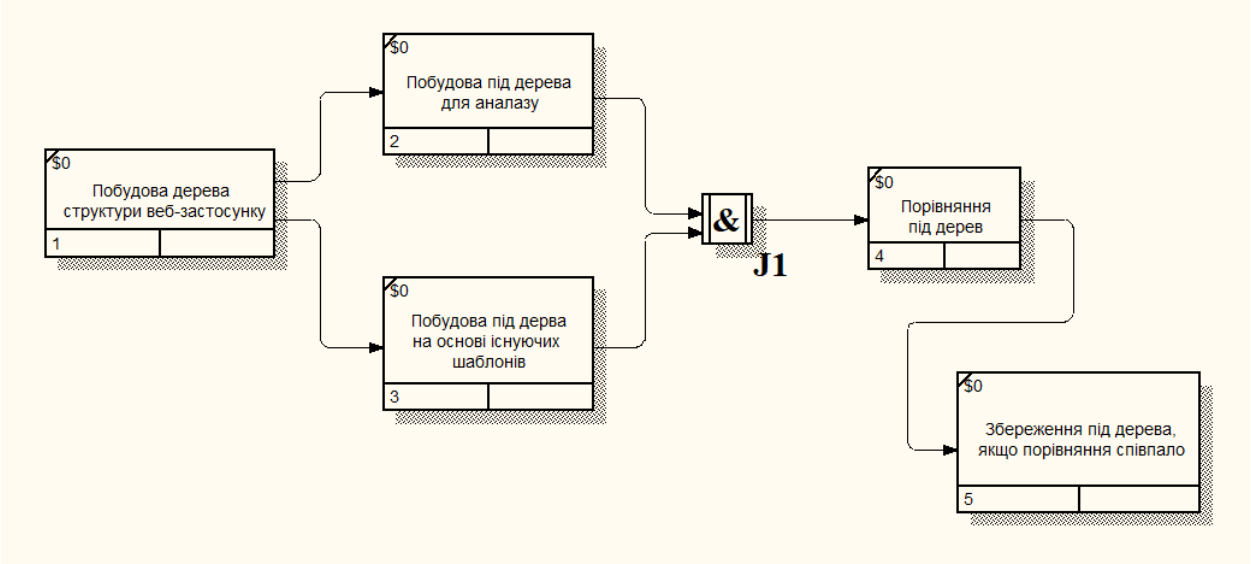


Рис. 2.4. Зв’язки між етапами роботами процесів «Аналізування дерева структури веб-застосунку»

2.3. Визначення потоків даних між етапами аналізування структури веб-застосунків на основі React.js

Для завершення побудови концептуальної моделі нам залишилось визначити потік даних між етапами аналізування структури веб-застосунків на основі React.js. Для цього ми використовували методику графічного структурного аналізу – DFD [11].

Діаграма даних (DFD) описує потік інформації для будь-якого процесу або системи. Він використовує певні символи, як прямокутники, кола та стрілки, а також короткі текстові мітки, щоб відображати дані вводу, виходи, точки збереження та маршрути між кожним пунктом призначення. Блоки даних діапазону можуть варіюватися від простих, навіть ручних процесів огляду, до глибинних, багаторівневих DFD, які копають поступово глибше в тому, як дані обробляються. Вони можуть бути використані для аналізу існуючої системи або моделювання нової. Як і всі найкращі діаграми та діаграми, DFD часто може візуально "сказати" речі, які було б важко пояснити словами, і вони працюють як на технічну, так і на нетехнічну аудиторію, від розробника до генерального директора. Ось чому DFD залишається настільки популярним після всіх цих років [11].

На рис. 2.5. зображено контекстна модель аналізування структури веб-застосунків на основі React.js в нотації DFD, яка вказує на те, що наш процес має 2 зовнішні об’єкти та використовує сховище шаблонів [8].

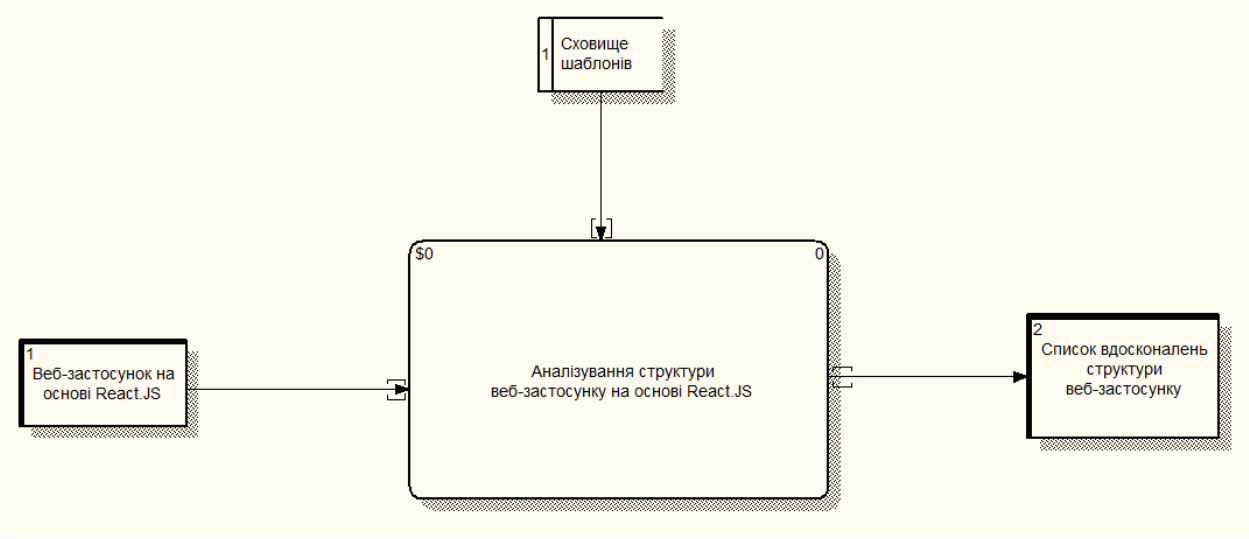


Рис. 2.5. Модель потоків даних програмного забезпечення для аналізування структури веб-застосунків на основі React.js в нотації DFD.

На рис. 2.6 зображена декомпозиція контекстної діаграми, яка відображає під процеси та потік даних між ними. Діаграма має наступні функції:

– здійснення парсингу структури веб-застосунку;
– аналізування структури веб-застосунку;
– створення списку рекомендацій.

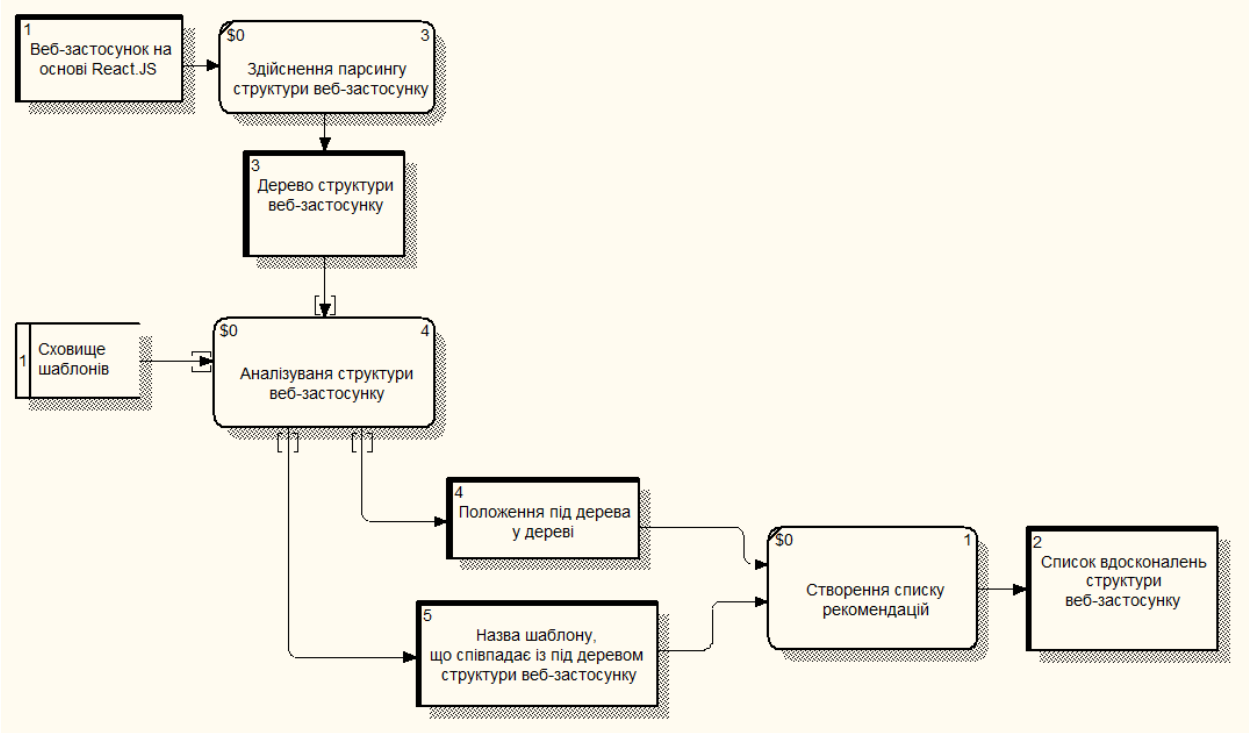


Рис. 2.6. Декомпозиція моделі потоків даних

Веб-застосунок на основі React.js надходить до процесу «Здійснення парсингу структури веб-застосунку». Результатом даного процесу є дерево структури веб-застосунку, над яким потім виконується процес «Аналізування структури веб-застосунку». Для виконання цього процесу необхідно використовувати сховище шаблонів. Після виконання процесу аналізування ми отримуємо інформацію щодо положення під дерева у дереві структури веб-застосунку, яке не задовольняє існуючим шаблонам, та інформацію щодо шаблону, завдяки якому було знайдено помилку у під дереві, що аналізується.

Ця інформація надходить до процесу «Створення списку рекомендацій», де відбувається форматування отриманої інформації та на виході синтезується список вдосконалень структури веб-застосунку.

На рис. 2.7. зображена схема структури потоків даних функціонального блоку «Аналізування структури веб-застосунку».

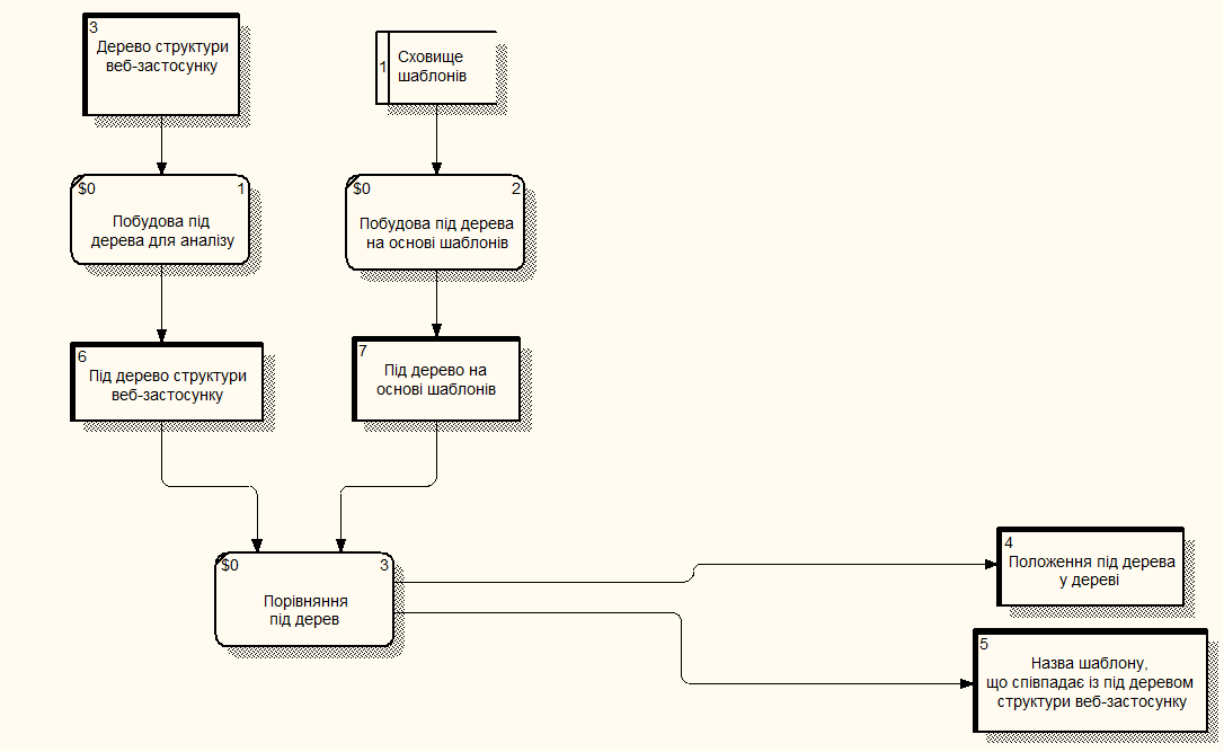


Рис. 2.7. – Декомпозиція функціонального блоку «Аналізування структури веб-застосунку»

Функції, зображені на даній схемі, виконуються циклічно, проходячи через кожну вершину дерева структури веб-застосунку, яке ми отримали на вході, та виконуючі зображені дії над кожною вершиною.

Спочатку будується під дерево таким чином, щоб його структура задовольняла вимоги аналізу. В той самий час будуються дерева на основі шаблонів проектування, що були взяті зі сховища шаблонів. Шаблони відображають корпоративні правила, яких необхідно дотримуватись при

розробці веб-застосунку, що аналізується. Ці шаблони створюються працівниками компанії, що використовує розроблений програмний засіб.

Кожна вершина під дерев містить необхідний набір даних для виконання їх порівнянь. У випадку, коли під дерева співпадають за визначеними критеріями, процес «Порівняння під дерев» зберігає положення під дерева у дереві структури веб-застосунку та інформацію щодо шаблону, завдяки якому було знайдено помилку.

2.4. Висновки

Для спрощення розуміння предметної області та завдань, які ставляться перед розроблюваним програмним засобом, у цьому розділі здійснено функціональне моделювання та графічний опис процесів в нотації IDEF0 з метою його формалізації та опису шляхом визначення функцій та під функцій, з яких складається основний процес.

Для визначення зв’язків між етапами роботи програмного засобу використано методику функціонального моделювання та графічного опису процесів IDEF3, який представляє механізм документування та збору інформації щодо послідовності виконання функцій під час процесу аналізування структури веб-застосунків на основі React.js.

Для визначення потоків даних програмного засобу аналізування структури веб-застосунків на основі React.js було побудовано діаграму в нотації DFD, що описує зовнішні, джерела даних, логічні функції, потоки даних і сховища даних, до яких здійснюється доступ.

Концептуальне моделювання дозволило формалізувати роботу програмного засобу набором функцій. Тоді як завдяки побудові процесної моделі та моделі потоків даних описано взаємозв’язки між етапами

аналізування структури веб-застосунків на основі React.js та обмін даних між ними. Зокрема це дозволить побудувати структуру програмного засобу

Основні результати розділу опубліковано в роботі [8].

3 ФІЗИЧНА МОДЕЛЬ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АНАЛІЗУВАННЯ СТРУКТУРИ ВЕБ-ЗАСТОСУНКІВ НА ОСНОВІ REACT.JS

3.1. Визначення варіантів використання програмного засобу аналізування структури веб-застосунків на основі React.js

Після завершення формалізування концептуальної моделі процесу аналізування структури веб-застосунків на основі React.js у попередньому розділі, ми можемо розпочати моделювання структури програмного засобу. І почнемо ми з визначення варіантів використання, які можна представити за допомогою діаграми прецедентів [12, 13].

Діаграма прецедентів - це представлення взаємодії користувача з системою, що показує взаємозв'язок між користувачем та різними випадками використання, в яких задіяний користувач. Діаграма прецедентів може визначати різні типи користувачів системи та різні випадки використання, а також часто супроводжуються іншими типами діаграм. Приклади використання представлені або кружками, або еліпсами.

У той час як сам корисний приклад може спричинити багато деталей щодо кожної можливості, схема користування може допомогти забезпечити вигляд системи на вищому рівні. Раніше було сказано, що "Використовувати діаграми прецедентів є кресленням для вашої системи" [12]. Вони надають спрощену та графічну інформацію про те, що система повинна реально виконувати.

Через їх спрощеність використання діаграми випадків може бути хорошим інструментом комунікації для зацікавлених сторін. Ці малюнки намагаються імітувати реальний світ і дають представникам зацікавлених сторін змогу зрозуміти, як система буде розроблена. Сіау та Лі провели дослідження, щоб визначити, чи існує дійсна ситуація для діаграм

прецедентів, або якщо вони були непотрібними. Результати дослідження показали, що схеми прецедентів передавали цілі системи більш спрощеною схемі для зацікавлених сторін [13].

Мета діаграми прецедентів полягає в тому, щоб забезпечити високий рівень перегляду системи. Додаткові діаграми та документацію можна використовувати для забезпечення повного функціонального та технічного перегляду системи [14].

Для проектування варіантів використання програмного засобу, що розроблюється в рамках магістерської дисертації, необхідно визначити акторів. Оскільки програмний засіб аналізування структури веб-застосунків на основі React.js ставить на меті дати можливість користувачеві аналізувати структуру веб-застосунків, єдиним учасником системи є розробник.

На рис. 3.1 зображено структурну схему варіантів використання відповідно до сформованих функціональних вимог [8].

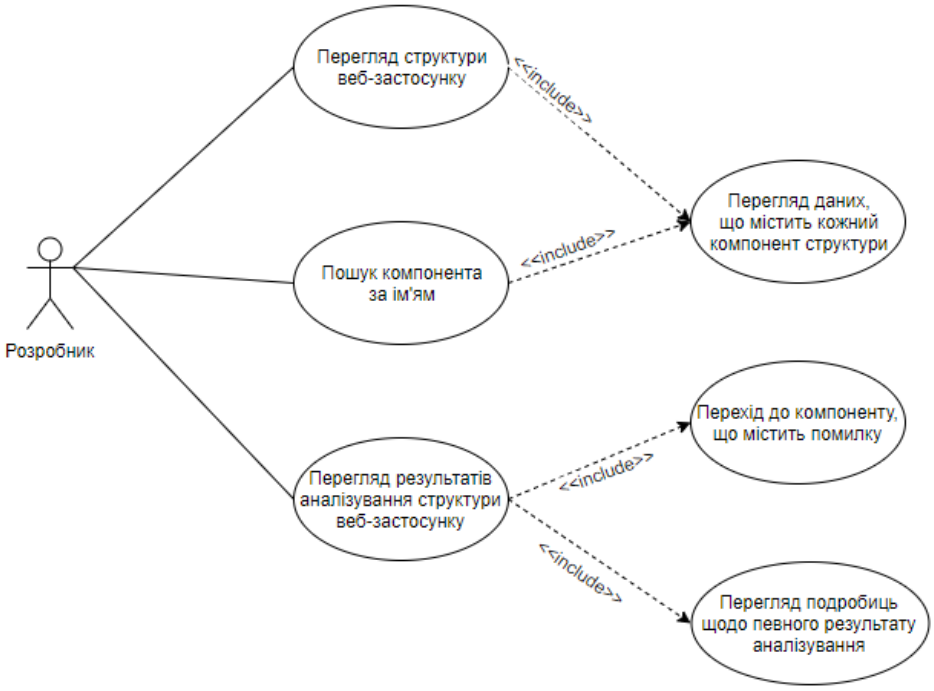


Рис. 3.1. Варіанти використання програмного забезпечення для аналізування структури веб-застосунків на основі React.js

Детальний опис кожного з варіантів використання наведено в табл. 3.1 – 3.6 [8].

Таблиця 3.1

Варіант використання «Перегляд структури веб-застосунку»

Найменування	Перегляд структури веб-застосунків
Первинний актор	Розробник
Інші актори	Немає
Опис	Можливість графічно переглядати структуру веб-застосунку
Попередні умови	Відкрита сторінка програмного засобу аналізування структури веб-застосунків на основі React.js, перед запуском програмного засобу було обрано правильний шлях до папки веб-застосунку
Вихідні умови	На сторінці програмного засобу аналізування структури веб-застосунків на основі React.js графічно зображено структуру веб-застосунку

Таблиця 3.2

Варіант використання «Пошук компонента за ім’ям»

Найменування	Пошук компонента за ім’ям
Первинний актор	Розробник
Інші актори	Немає
Опис	Можливість пошуку компоненту за ім’ям в рамках структури веб-застосунку, що аналізується
Попередні умови	Відкрита сторінка програмного засобу аналізування структури веб-застосунків на основі React.js, перед його запуском обрано правильний шлях до папки веб-застосунку, що аналізується, у формі пошуку введено ім’я компоненту

Продовження табл. 3.2

Вихідні умови	На сторінці програмного засобу аналізування структури веб-застосунків на основі React.js на місці, де було зображено структуру веб-застосунку, графічно зображено список компонентів, назва котрих задовольняє пошук застосунку, графічно зображено список компонентів, назва котрих задовольняє пошук

Таблиця 3.3

Варіант використання «Перегляд результатів аналізування структури веб-застосунку»

Найменування	Перегляд результатів аналізування структури веб-застосунку
Первинний актор	Розробник
Інші актори	Немає
Опис	Можливість перегляду результатів аналізування структури веб-застосунку, тобто список покращень структури веб-застосунку у випадку, якщо такі покращення були знайдені
Попередні умови	Відкрита сторінка програмного засобу аналізування структури веб-застосунків на основі React.js, перед його запуском обрано правильний шлях до папки веб-застосунку, що аналізується, обрана вкладка «Results» зверху від графічного зображення структури веб-застосунку
Вихідні умови	На сторінці програмного засобу аналізування структури веб-застосунків на основі React.js на місці, де було зображено структуру веб-застосунку, графічно зображено список покращень

Таблиця 3.4.

Варіант використання «Перегляд даних, що містить кожний компонент структури»

Найменування	Перегляд даних, що містить кожний компонент структури
Первинний актор	Розробник
Інші актори	Немає
Опис	Можливість перегляду даних, що містить кожний компонент структури веб-застосунку, а саме тих даних, що використовуються для аналізування структури
Попередні умови	Відкрита сторінка програмного засобу аналізування структури веб-застосунків на основі React.js, перед запуском програмного засобу було обрано правильний шлях до папки веб-застосунку, обрано компонент, дані якого необхідно переглянути
Вихідні умови	На сторінці програмного засобу аналізування структури веб-застосунків на основі React.js праворуч від графічного зображення структури веб-застосунку зображено дані, які містить в собі обраний компонент

Таблиця 3.5

Варіант використання «Перехід до компоненту, що містить помилку»

Найменування	Перехід до компоненту, що містить помилку
Первинний актор	Розробник
Інші актори	Немає
Опис	Можливість переходу до компоненту, що містить помилку, для подальшого перегляду його даних та пошуку вирішення конкретного випадку

Продовження табл. 3.5

Попередні умови	Відкрита сторінка програмного засобу аналізування структури веб-застосунків на основі React.js, перед запуском програмного засобу було обрано правильний шлях до папки веб-застосунку, що аналізується, обрана вкладка «Results» зверху від графічного зображення структури веб-застосунку, обрано конкретний пункт зі списку покращень
Вихідні умови	На сторінці програмного засобу аналізування структури веб-застосунків на основі React.js на місці, де було зображено структуру веб-застосунку, графічно зображено компонент, що належить на обраного покращення

Таблиця 3.6

Варіант використання «Перегляд подробиць щодо певного результату аналізування»

Найменування	Перегляд подробиць щодо певного результату аналізування
Первинний актор	Розробник
Інші актори	Немає
Опис	Можливість перегляду подробиць щодо певного результату аналізування
Попередні умови	Відкрита сторінка програмного засобу аналізування структури веб-застосунків на основі React.js, перед запуском програмного засобу було обрано правильний шлях до папки веб-застосунку, що аналізується, обрана вкладка «Results» зверху від графічного зображення структури веб-застосунку, обрано конкретний пункт зі списку покращень

Продовження табл. 3.6

Вихідні умови	На сторінці програмного засобу аналізування структури веб-застосунків на основі React.js на місці, де було зображено структуру веб-застосунку, зображено назву та пояснення обраного покращення

3.2. Визначення логічної структури програмного засобу аналізування структури веб-застосунків на основі React.js

На основі концептуальної модель, описаної у розділі 2, та схеми прецедентів, описаної у попередньому підрозділі, ми можемо побудувати логічну структуру програмного засобу аналізування структури веб-застосунків на основі React.js [8].

Логічна архітектура підтримує функціонування системи протягом усього її життєвого циклу на логічному рівні. Вона складається з набору пов'язаних технічних концепцій і принципів. Логічна архітектура представляється за допомогою методів, що відповідають тематичними групами описів, і як мінімум, включає в себе функціональну архітектуру, поведінкову архітектуру і тимчасову архітектуру [15].

Логічна структура містить набір функціонально-логічних модулів, що включають процедури і об'єкти, що представляють собою стандартні прототипи додатків баз даних: форми, вікна для перегляду таблиць бази даних, звіти запити тощо і оригінальні програмні одиниці, що реалізують деяку автоматизуються функцій або завдання досліджуваної предметної області.

Базовим принципом методології структурного підходу є принцип декомпозиції, згідно з яким при проектуванні програмного забезпечення

здійснюється його функціональна декомпозиція на відповідні підсистеми і модулі, що виконують певні функції [16].

На рис. 3.2 зображено структурну схему компонентів програмного засобу аналізування структури веб-застосунків на основі React.js. Опис кожного компоненту, що використовується у програмному засобі, наведено в таблиці 3.7. Список основних методів компонентів, описаних в таблиці 3.7. наведено в таблицях 3.8.-3.13.

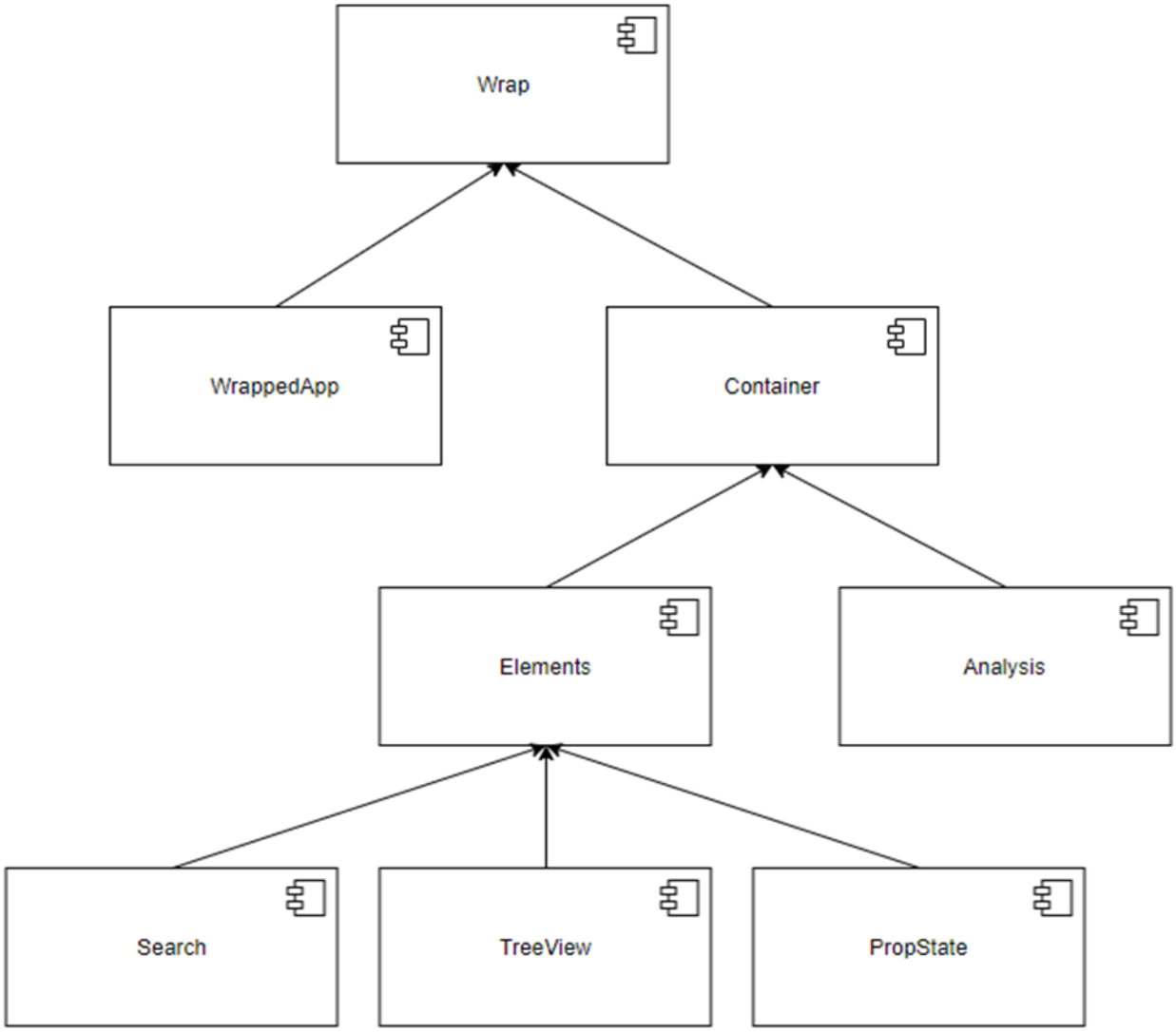


Рис. 3.2. Структурна схема компонентів програмного забезпечення для аналізування веб-застосунків на основі React.js

Таблиця 3.7.

Список компонентів програмного засобу

Компонент	Опис
Wrap	Компонент має структурну функцію та призначений для відображення веб-застосунку, що аналізується, парсингу його структури та допоміжної панелі, де зображена вся додаткова інформація
WrappedApp	Компонент призначений для відображення веб-застосунку, що аналізується
Container	Компонент має структурну функцію та призначений регулювати, яку допоміжну панель відображати
Elements	Компонент має структурну функцію, містить в собі всі необхідні компоненти для відображення структури веб-застосунку, що аналізується, пошуку компонентів за ім’ям, відображення даних, що знаходяться в певному компоненті
Analysis	Компонент призначений для аналізування структури веб-застосунку та для відображення результатів аналізу
Search	Компонент призначений для пошуку компонентів за ім’ям всередині структури веб-застосунку, що аналізується
TreeView	Компонент призначений для відображення структури веб-застосунку, що аналізується
PropState	Компонент призначений для відображення даних, що знаходяться в обраному користувачем компоненті

Таблиця 3.8

Список основних методів компоненту Wrap

Метод	Опис
render	Метод відповідає за відображення веб-застосунку (компонент WrappedApp), що аналізується, та допоміжної панелі (компонент Container)

Таблиця 3.9

Список основних методів компоненту Container

Метод	Опис
render	Метод відповідає за відображення обраної (активної) вкладинки (компоненти Elements та Analysis)
setSelectedTab	Метод відповідає за зміну активної вкладинки, що відображається за допомогою методу render

Таблиця 3.10

Список основних методів компоненту Elements

Метод	Опис
render	Метод відповідає за відображення структури веб-застосунку (компонент TreeView), пошукової панелі (компонент Search) та даних, що знаходяться в обраному компоненті веб-застосунку (компонент PropState)

Таблиця 3.11

Список основних методів компоненту TreeView

Метод	Опис
render	Метод відповідає за відображення структури веб-застосунку, що аналізується
setRoot	Метод встановлює нову голову дерева, що відображає під дерево структури веб-застосунку. Використовується для відображення результатів пошуку компонентів за ім’ям

Таблиця 3.12

Список основних методів компоненту Search

Метод	Опис
render	Метод відповідає за відображення пошукової панелі
searchNode	Метод відповідає за пошук компоненту за ім’ям. Повертає масив компонентів, що відповідають введеному значенню, чи пустий масив у випадку, якщо компонентів з таким ім’ям не було знайдено. Встановлює знайді компоненти головами дерев за допомогою методу setRoot компоненту TreeView
clearSearch	Метод відповідає за очищення результатів пошуку та повертає голову дерева структури веб-застосунку

Таблиця 3.13.

Список основних методів компоненту PropState

Метод	Опис
render	Метод відповідає за відображення даних в обраному користувачем компоненті

Продовження табл. 3.13

getData	Метод відповідає за отримання даних, що знаходяться в обраному користувачем компоненті

3.3. Визначення фізичної структури програмного забезпечення для аналізування структури веб-застосунків на основі React.js

На основі логічної моделі, описаної у попередньому підрозділі, було розроблено фізичну структуру програмного засобу аналізування структури веб-застосунків на основі React.js. Діаграма компонентів програмного засобу зображена на рис. 3.3. та містить компоненти, що описані в попередньому підрозділі, не включаючи головного компоненту, що відображає веб-застосунок, що аналізується [8]:

- TabbedPane – компонент, що визначає , яку вкладинку відображати, та забезпечує можливість перемикати вкладинки;
- Elements – вкладинка, що містить компоненти для відображення структури веб-застосунку, що аналізується, пошукову панель та дані про обраний компонент структури;
- SplitPane – компонент, який має структурну функцію, відповідає за місце відображення структури веб-застосунку та даних про її обраний компонент;
- TreeView – компонент, який відображає структуру веб-застосунку, що аналізується;
- Search – компонент, який відповідає за пошук компоненту за ім’ям в рамках структури веб-застосунку, що аналізується;
- PropState – компонент, який відображає дані про обраний користувачем компонент структури;

– Analysis – компонент, який відповідає за аналізування структури веб-застосунку та відображення результатів аналізу.

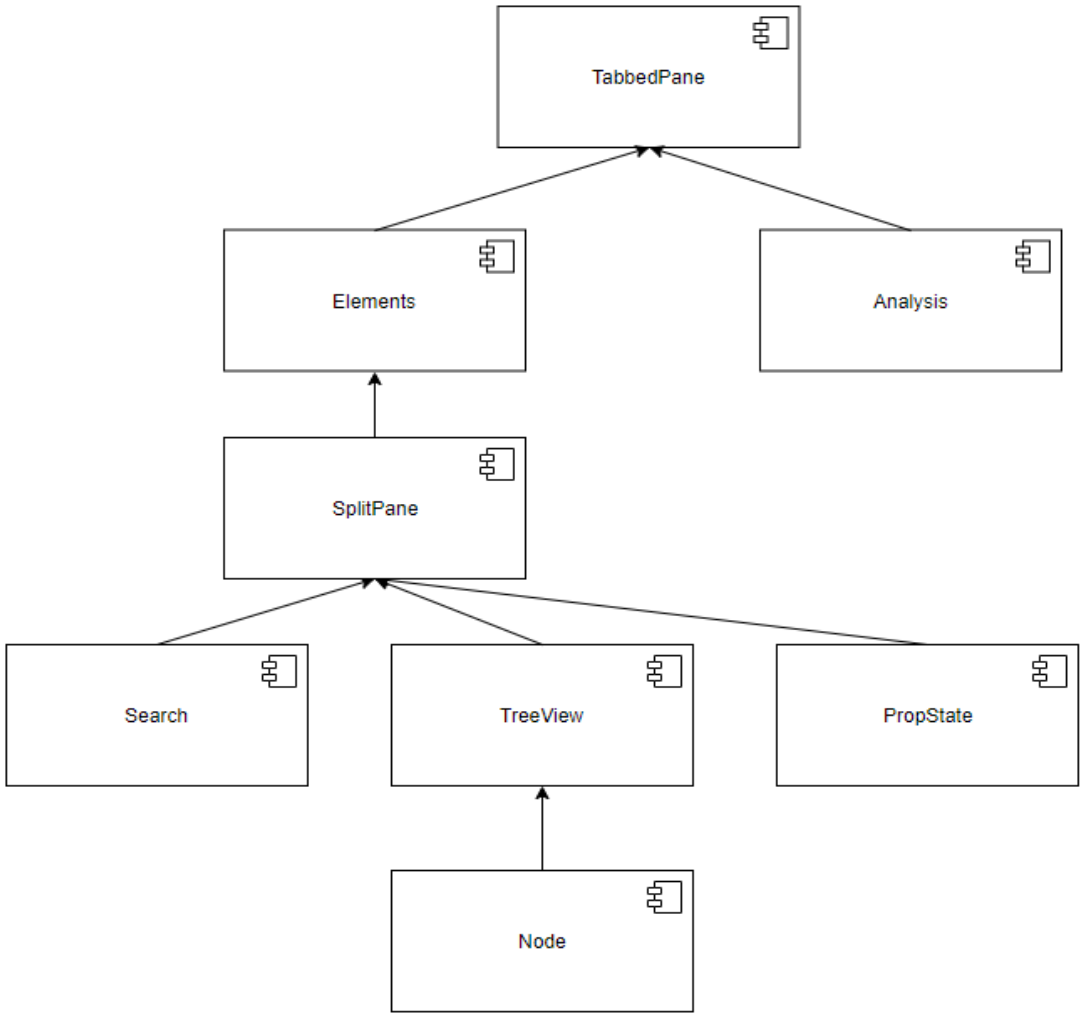


Рис. 3.3. Фізична модель програмного забезпечення для аналізування структури веб-застосунків на основі React.js

3.4. Висновки

На основі концептуальної моделі, розробленої та описаної у розділі 2, визначено варіанти використання та побудовано фізичну модель програмного забезпечення для аналізування структури веб-застосунків на основі React.js.

Для проектування варіантів використання визначено дійових осіб та використано UML-діаграму прецедентів.

Логічна структура містить набір функціонально-логічних компонентів, що включають процедури і об'єкти, що представляють собою стандартні прототипи додатків баз даних та відображена у вигляді відповідної UML-діаграми.

Для побудови фізичної моделі програмного забезпечення для аналізування структури веб-застосунків на основі Rect.js було використано діаграму компонентів.

Основні результати розділу опубліковано в роботі [8].

4 РОБОЧИЙ ПРОЕКТ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АНАЛІЗУВАННЯ СТРУКТУРИ ВЕБ-ЗАСТОСУНКІВ НА ОСНОВІ REACT.JS

4.1. Реалізування програмного забезпечення для аналізування структури веб-застосунків на основі React.js

Огляд предметної області та визначення функціональних вимог, що описані у розділі 1, формалізація концептуальної моделі процесу аналізування структури веб-застосунків на основі React.js, а саме визначення зв’язків між його етапами та визначення потоку даних між етапами, описані в розділі 2, визначення варіантів використання, що дало змогу побудувати логічну та фізичні структури програмного засобу у розділі 3, були необхідними етапами для реалізації програмного засобу аналізування структури веб-застосунків на основі React.js. Саме за допомогою цих складових було прийнято рішення стосовно архітектури програмного засобу, технологій та шаблонів, які були використанні при його розробці [8].

Оскільки з самого початку метою магістерської дисертації був синтез програмного засобу, який будуть використовувати розробники, то і технології обирались такими, що будуть добре знайомими для майбутніх користувачів програмним засобом. Таким чином, через аналізування структури веб-застосунків саме на основі React.js, що використовує мову програмування Javascript як основну, мовою програмування для програмного засобу було обрано також Javascript та React.js для реалізації графічного інтерфейсу. Для побудови дерева структури веб-застосунку, що аналізується, було використано пакет React-devtools від розробників бібліотеки React.js.

JavaScript спочатку створювався для того, щоб зробити web-сторінки «живими». Програми на цій мові програмування називаються скриптами. У

браузері вони підключаються безпосередньо до HTML і, як тільки завантажується сторінка - тут же виконуються.

Сучасний JavaScript - це «безпечна» мова програмування загального призначення. Вона не надає низькорівневих засобів роботи з пам'яттю, процесором, так як спочатку була орієнтована тільки на браузери, в яких це не потрібно [17].

Що ж стосується інших можливостей - вони залежать від оточення, в якому запущений JavaScript. У браузері JavaScript вміє робити все, що відноситься до маніпуляції зі сторінкою, взаємодії з відвідувачем і, в якійсь мірі, з сервером:

- створювати нові HTML-теги, видаляти існуючі, змінювати стилі елементів, ховати, показувати елементи і тощо;
- реагувати на дії відвідувача, обробляти кліки миші, переміщення курсора, натискання на клавіатуру і тощо;
- Посилати запити на сервер і завантажувати дані без перезавантаження сторінки (ця технологія називається "AJAX") [18, 19].

JavaScript - швидка та потужна мова, але браузер накладає на її виконання деякі обмеження. Це зроблено для безпеки користувачів, щоб зловмисник не міг за допомогою JavaScript отримати особисті дані або якось нашкодити комп'ютеру користувача. Хоча, цих обмежень немає там, де JavaScript використовується поза браузера, наприклад на сервері. Крім того, сучасні браузери надають свої механізми по установці плагінів і розширень, які володіють розширеними можливостями, але вимагають спеціальних дій по установці від користувача [20].

React - це декларативна, ефективна та гнучка бібліотека JavaScript для побудови користувацьких інтерфейсів. Це дозволяє створювати складні

інтерфейси користувача з невеликих і ізольованих частин коду, які називаються "компонентами" [21].

З метою полегшити процес розробки веб-застосунків, було розроблено формат JSX, що поставляється з повною потужністю JavaScript. Ви можете виставити будь-які вирази JavaScript у дужках всередині JSX. Кожен реактивний елемент є об'єктом JavaScript, який ви можете зберігати в змінній або передавати в своїй програмі [22].

Переваги використання JSX:

- вкладені елементи. Кілька елементів на одному рівні повинні бути загорнуті в один елемент контейнера, наприклад елемент , показаний вище, або повернений як масив [23];
- атрибути. JSX надає ряд атрибутів елементів, призначених для відображення тих, що надаються у форматі HTML. Користувальницькі атрибути також можуть бути передані компоненту.[24] Всі атрибути будуть отримані компонентом як параметри;
- вирази Javascript. Вирази JavaScript (але не твердження) можуть використовуватися всередині JSX з фігурними дужками {};
- умовні твердження. if-else твердження не можуть бути використані всередині JSX, але замість них можуть використовуватися умовні вирази.[25]

Отже, обраний набір технологій для реалізації програмного засобу в рамках магістерської дисертації має наступні переваги:

- програмний засіб та веб-застосунок, що аналізується, написані за допомогою однакового набору технологій, тобто користувачі, розробники веб-застосунку, не матимуть проблем з використанням програмного засобу і матимуть можливість

швидко зрозуміти принципи роботи та, у разі необхідності, відредагувати програмний засіб під особисті вимоги;

– обраний набір технологій для реалізації програмного засобу є дуже популярним у світі веб-розробки, тому має велику спільноту розробників. Отже, користувач, що не має досвіду роботи у програмуванні, матиме можливість швидко та легко знайти допомогу та відповіді на свої запитання щодо реалізації програмного засобу у спільноти розробників у мережі інтернету;
– програмний засіб реалізований у вигляді веб-застосунку. Тому використання не потребує особливих умов для роботи, достатньо будь-якого браузера. І його буде легко інтегрувати у проект, що описаний у розділі 5.

4.2. Тестування програмного забезпечення для аналізування структури веб-застосунків на основі React.js

Тестування програмного забезпечення - це дослідження, проведене для надання зацікавленим сторонам інформації про якість випробуваного програмного продукту чи послуги [26]. Тестування програмного забезпечення також може забезпечити об'єктивний, незалежний перегляд програмного забезпечення, що дозволяє підприємству оцінити та зрозуміти ризики впровадження програмного забезпечення. Методи тестування включають в себе процес виконання програми з метою виявлення програмних помилок (помилок або інших дефектів) та перевірки того, що програмний продукт підходить для використання.

Тестування програмного забезпечення передбачає виконання програмного компонента для оцінки одного або декількох цікавих

властивостей. Загалом, ці властивості вказують на те, наскільки випробуваний компонент чи система:

- відповідає вимогам, які керують його розробкою та розробкою;
- правильно відповідає на всі види входів;
- виконує свої функції протягом прийнятного часу;
- достатньо корисний;
- може бути встановлений і запускатися в передбачуваних середовищах;
- досягає загального результату, якого зацікавлені сторони очікують.

Оскільки кількість можливих тестів для рівних простих програмних компонентів практично нескінченна, у всіх програмних тестах використовується певна стратегія вибору тестів, які є можливими для наявного часу та ресурсів.

Тестування програм можна проводити, як тільки виконуване програмне забезпечення (навіть якщо це частково завершено) існує. Загальний підхід до розробки програмного забезпечення часто визначає, коли і як проводиться тестування. Наприклад, в поетапному процесі більшість випробувань відбувається після того, як системні вимоги були визначені, а потім реалізовані в тестованих програмах. Навпаки, під гнучким підходом, вимоги, програмування та тестування часто виконуються одночасно [27].

Після реалізації програмного засобу аналізування структури веб-застосунків на основі React.js, отриманий засіб повинен пройти етап тестування для перевірки функціональних вимог, описаних в підрозділі 1.3. В таблицях 4.1-4.4 наведено інформацію про тестування варіантів використання та їх результати зображені на рис. 4.2 та 4.3 [8].

Таблиця 4.1

Перевірка перегляду структури веб-застосунку

Мета тесту	Перевірка можливості перегляду структури веб-застосунку
Початковий стан	Було обрано правильний шлях до папки веб-застосунку, що аналізується, відкрито сторінку програмного засобу
Вхідні дані	Немає
Схема проведення тесту	Після відкриття сторінки програмного засобу у правій частині екрану побачити структуру веб-застосунку, що аналізується
Очікуваний результат	В правій частині екрану побачили структуру веб-застосунку, що аналізується
Стан програмного продукту після проведення випробувань	В правій частині екрану побачили структуру веб-застосунку, що аналізується

Таблиця 4.2

Перевірка пошуку компонента за ім’ям

Мета тесту	Перевірка можливості пошуку компонента за ім’ям
Початковий стан	Було обрано правильний шлях до папки веб-застосунку, що аналізується, відкрито сторінку програмного засобу
Вхідні дані	Ім’я компонента, що необхідно знайти
Схема проведення тесту	В правій частині екрану, в пошуковій панелі вводимо ім’я компонента, що необхідно знайти

Продовження табл. 4.2

Очікуваний результат	На місці, де було розташовано дерево структури веб-застосунку, що аналізується, відображено під дерева, головними вершинами котрих є компоненти, імена яких задовольняють пошуку
Стан програмного продукту після проведення випробувань	На місці, де було розташовано дерево структури веб-застосунку, що аналізується, відображено під дерева, головними вершинами котрих є компоненти, імена яких задовольняють пошуку

Таблиця 4.3

Перевірка перегляду результатів аналізування структури веб-застосунку

Мета тесту	Перевірка можливості перегляду результатів аналізування структури веб-застосунку
Початковий стан	Було обрано правильний шлях до папки веб-застосунку, що аналізується, відкрито сторінку програмного засобу
Вхідні дані	Немає
Схема проведення тесту	Після відкриття сторінки програмного засобу в правій частині екрану обираємо вкладинку «Results» та на місці, де було зображено дерево структури веб-застосунку, що аналізується, можна побачити результати аналізування структури веб-застосунку
Очікуваний результат	На місці, де було зображено дерево структури веб-застосунку, що аналізується, зображено результати аналізування структури веб-застосунку

Продовження табл. 4.3

Стан програмного продукту після проведення випробувань	На місці, де було зображено дерево структури веб-застосунку, що аналізується, зображено результати аналізування структури веб-застосунку

Таблиця 4.4

Перевірка перегляду даних, що містить кожний компонент структури

Мета тесту	Перевірка можливості перегляду даних, що містить кожний компонент структури
Початковий стан	Було обрано правильний шлях до папки веб-застосунку, що аналізується, відкрито сторінку програмного засобу
Вхідні дані	Немає
Схема проведення тесту	Після відкриття сторінки програмного засобу обираємо певний компонент із структури і праворуч від місця, де зображено структуру веб-застосунку, можна побачити дані про обраний компонент
Очікуваний результат	Праворуч від місця, де зображено структуру веб-застосунку, зображено дані про обраний компонент
Стан програмного продукту після проведення випробувань	Праворуч від місця, де зображено структуру веб-застосунку, зображено дані про обраний компонент

Таблиця 4.5

Перевірка переходу до компонента, що містить помилку

Мета тесту	Перевірка можливості переходу до компонента, що містить помилку

Продовження табл. 4.5

Початковий стан	Було обрано правильний шлях до папки веб-застосунку, що аналізується, відкрито сторінку програмного засобу
Вхідні дані	Немає
Схема проведення тесту	Після відкриття сторінки програмного засобу в правій частині екрану обираємо вкладинку «Results» та на місці, де було зображено дерево структури веб-застосунку, що аналізується, обираємо один із результатів аналізування структури веб-застосунку
Очікуваний результат	На місці, де було зображено результати аналізування структури веб-застосунків, зображено під дерево структури, головною вершиною якого є компонент, який відповідає обраному результату
Стан програмного продукту після проведення випробувань	На місці, де було зображено результати аналізування структури веб-застосунків, зображено під дерево структури, головною вершиною якого є компонент, який відповідає обраному результату

4.3. Використання програмного забезпечення для аналізування структури веб-застосунків на основі React.js

Програмний засіб аналізування структури веб-застосунків на основі React.js було реалізовано на основі архітектури «товстий клієнт».

Товстий клієнт в архітектурі клієнт-сервер або в мережах, що зазвичай забезпечує багату функціональність незалежно від центрального сервера. Спочатку він відомий як "клієнт" або "товстий клієнт" [28], назва протиставляється тонкому клієнту, який описує комп'ютер, сильно залежний від додатків сервера. Товстий клієнт все-таки вимагає щонайменше періодичного підключення до мережі або центрального сервера, але часто

характеризується здатністю виконувати безліч функцій без цього з'єднання. На відміну від цього, тонкий клієнт, як правило, робить якнайменше обробку, покладаючись на доступ до сервера кожного разу, коли вхідні дані потрібно обробляти або перевірити.

Переваги використання даної архітектури:

- Низькі вимоги до сервера. Товстий клієнт-сервер не вимагає такого високого рівня продуктивності, як тонкий клієнт-сервер (оскільки товсті клієнти самі роблять велику частину обробки додатків). Це призводить до істотно дешевих серверів;
- Робота в режимі офлайн. Товсті клієнти мають переваги в тому, що постійне з'єднання з центральним сервером часто не вимагається;
- Краща мультимедійна продуктивність. Товсті клієнти мають переваги в багатих мультимедіа додатках, що буде інтенсивною пропускною здатністю, якщо вона буде повністю обслуговуватися. Наприклад, товсті клієнти добре підходять для відеоігор;
- Більша гнучкість. На деяких операційних системах програмні продукти призначені для персональних комп'ютерів, які мають власні локальні ресурси. Запуск цього програмного забезпечення у тонкому клієнтському середовищі може бути складним;
- Використання існуючої інфраструктури. Оскільки багато людей зараз мають дуже швидкі локальні ПК, у них вже є інфраструктура для запуску товстих клієнтів без додаткової вартості;
- Вища ємність сервера. Чим більше роботи виконується клієнтом, тим менше серверу потрібно зробити, збільшуючи кількість користувачів, що сервер може підтримувати [29].

Для встановлення програмного засобу необхідно мати встановленими на комп’ютері наступні пакети:

1. Node.js [30];
2. WebStorm [31];
3. Git [32].

Для збору програмного засобу необхідно ввести наступні команди, знаходячись в кореневій папці програмного засобу:

1. npm i;
2. ./shell/plain/build.sh;
3. ./test/example/build.sh.

Для запуску програмного засобу необхідно відрити файл ./shell/plain/index.js за допомогою додатку WebStorm.

4.4. Висновки

У цьому розділі розглянуто набір технологій, що використовувався для реалізації програмного засобу аналізування структури веб-застосунків на основі React.js, переваги використання деяких з них. Було протестовано варіанти використання на наявність функціональних вимог.

Також було розглянуто метод розгортання та запуску програмного засобу на власному комп’ютері.

Основні результати розділу опубліковано в роботі [8].

5 СТАРТАП-ПРОЕКТ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АНАЛІЗУВАННЯ СТРУКТУРИ ВЕБ-ЗАСТОСУНКІВ НА ОСНОВІ REACT.JS

5.1. Опис ідеї проекту

На сьогоднішній день більшість прикладних програм для комп'ютерних програм є веб-додатками, використання веб-додатків зростає з кожним днем, тому фахівці з інформаційних технологій повинні знайомитися з веб-додатками, зокрема розробниками програмного забезпечення та програмними тестерами.

Веб-застосунок – це програмне забезпечення, яке працює на веб-сервері. На відміну від традиційних настільних додатків, які запускаються веб-браузером. Тому веб-застосунки зазвичай написані на мові, що підтримується браузером, такими як JavaScript та HTML, оскільки ці мови залежать від браузера, щоб зробити програму виконуваною. Деякі програми є динамічними, що потребують обробки на стороні сервера. Інші повністю статичні без обробки, необхідної на сервері.

Веб-застосунок вимагає наявності веб-сервера для керування запитами від клієнта, сервера додатків для виконання запитаних завдань та, іноді, бази даних для зберігання інформації. Технологія сервера додатків коливається від ASP.NET [33], ASP та ColdFusion [34], до PHP [35] та JSP [36].

Якщо це настільна програма, як Notepad [37] або Acrobat Reader [38], ми можемо запустити ці програми з операційної системи, просто увімкніть Комп'ютер, і після завантаження системи ми зможемо керувати робочими додатками. Якщо це веб-застосунок, завантажте систему, запустіть веб-переглядач та перейдіть до веб-додатки, введіть URL-адресу веб-програми на адресній панелі веб-переглядача, після чого відкрийте додаток, закрийте веб-переглядач на кінці [39].

Мільйони підприємств використовують Інтернет як економічно ефективний канал зв'язку. Це дозволяє їм обмінюватися інформацією з цільовим ринком та здійснювати швидкі, безпечні транзакції. Проте ефективне залучення можливе лише тоді, коли бізнес здатний захоплювати та зберігати всі необхідні дані, а також мати можливість обробляти цю інформацію та представляти результати користувачеві.

Веб-застосунки використовують комбінацію скриптів на стороні сервера (PHP та ASP) для обробки інформації про зберігання та вилучення інформації, а також для сценаріїв на стороні клієнта (JavaScript та HTML) для надання користувачам інформації. Це дозволяє користувачам взаємодіяти з компанією за допомогою онлайн форм, систем управління контентом, кошиків для магазинів тощо. Крім того, програми дозволяють працівникам створювати документи, обмінюватися інформацією, співпрацювати над проектами та працювати над загальними документами незалежно від місцезнаходження чи пристрою [40].

React.js є однією з таких бібліотек Javascript, яка досягла масової популярності в області інтернет-бізнесу, а також за цілком привабливою причиною. Єдине, що він робить виключно добре - вирізати чудові користувацькі інтерфейси (інтерфейс користувача). З принципом, що HTML та JavaScript обов'язково співпрацюють один з одним, React був створений з урахуванням бізнес-форм, використовуючи швидше швидкість завантаження веб-сторінки, зручність в SEO та повторне використання коду завдяки поєднанню двох технологій.

Підприємства швидко сприймали цю технологію через простоту, яку вона запропонувала розробникам; тобто здатність навчитися реагувати в мінімальний час. Повторне використання коду з бездоганним додаванням / модифікацією функціональних можливостей в існуючій системі означало

значно менше часу і бюджету на розробку та створення більших команд, відповідно [41].

Зазвичай, веб-застосунки, що використовують React.js, складаються з декількох сотень або тисяч компонентів, що створює додаткову проблему для розробників – складність запам’ятати структуру компонентів, призначення певних компонентів та імена компонентів для необхідного компоненту. Результатом такого безладу є те, що в одному проекті може бути декілька компонентів, що виконують однакові функціональні вимоги.

Для вирішення цієї проблеми компанії створюють внутрішні корпоративні правила, дотримання яких вирішує проблему з пошуком необхідних компоненті. Ці правила також створені для того, щоб розробники з самого початку мали можливість правильно називати компоненти.

На основі виявлених проблем було сформовано дерево проблем, яке зображено на рис. 5.1.

Рис. 5.1. Дерево проблем

5.2. Аналіз ринкових можливостей запуску стартап-проекту

Зацікавлені сторони – це особи або організації (наприклад, замовники, спонсори, виконуюча організація або громадськість), які активно беруть участь в проекті або інтереси яких можуть бути порушені як позитивно, так і негативно в ході виконання або в результаті завершення проекту. Зацікавлені сторони проекту також можуть впливати на проект, його результати і на членів команди проекту. Команда управління проектом повинна виявити як внутрішніх, так і зовнішніх зацікавлених сторін проекту, щоб визначити вимоги, що пред'являються до проекту, і очікування всіх залучених сторін. Крім того, менеджер проекту повинен управляти впливом різних зацікавлених сторін проекту в зв'язку з вимогами, що пред'являються до проекту, щоб забезпечити успішне отримання результату.

Зацікавлені сторони проекту мають різні ступені відповідальності і повноважень за участю в проекті, які можуть змінюватися протягом життєвого циклу проекту. Їх відповідальність і повноваження можуть варіюватися від періодичного участі в опитуваннях і цільових групах до повного спонсорства проекту, що включає надання фінансової та політичної підтримки. Зацікавлені сторони проекту можуть надавати несприятливий вплив на цілі проекту [42].

В рамках магістерської дисертації та безпосередньо програмного забезпечення для аналізування структури веб-застосунків на основі React.js було визначено наступні зацікавлені сторони:

– розробник. Оцінка роботи розробника залежить від багатьох акторів, одним з яких є якість коду, що він пише;

– проектний менеджер. Зацікавлений у використанні програмного забезпечення для аналізування структури веб-застосунків на основі React.js, так як саме він є крайньою людиною, яка відповідає за програмний засіб

перед клієнтом, адже якість продукту, який продає компанія, впливає на репутацію компанії;

– клієнт. Основною метою створення веб-застосунків може бути його подальше використання для продажу власних товарів чи послуг, мається на увазі використання даного веб-застосунку покупцями. Тому для клієнта важливо, щоб веб-застосунок був надійним.

Після визначення зацікавлених сторін необхідно провести аналіз та визначити, хто має найбільший вплив на реалізацію програмного забезпечення для аналізування структури веб-застосунків на основі React.js. Аналіз зацікавлених сторін зображено на табл. 5.1.

Таблиця 5.1

Аналіз зацікавлених сторін

Фактори	Можливості	Влада	Вплив		
			Влада	Інтерес	Сума
Розробники	Створюють програмне забезпечення, в нашому випадку веб-застосунки на основі React.js	Забезпечують готовий програмний засіб	2	2	4
Проектний менеджер	Забезпечують дотримання термінів виконання вимог щодо реалізації веб-застосунку	Має вплив на розробників та відповідні за переговори з клієнтом	5	7	35
Клієнт	Визначати вимоги до програмного засобу, термінів виконання	Має вплив на команду, що працює над веб-застосунком,	9	10	90

Споживачі:

- IT компанії;
- Юридична особа, якій необхідний веб-застосунок.

Проблема:

- Сучасні методи та програмні засоби нездатні аналізувати структуру веб-застосунків, так чином не можуть виявляти проблеми на рівні структури.

Рішення:

- Програмне забезпечення для аналізування структури веб-застосунків на основі React.js.

Унікальна ціннісна пропозиція:

- перевірка веб-застосунку один раз, з підтримкою налагодження;
- місячна підписка на 10 перевірок веб-застосунків;
- місячна підписка з безмежною кількістю перевірок веб-застосунків.

Потоки доходів:

- продаж можливості використання програмним засобом для аналізування структури веб-застосунків на основі React.js;
- продаж підписок на доступ використання програмного засобу з безмежною кількістю перевірок;
- продаж підписок на доступ використання програмного засобу з обмеженою кількістю перевірок.

Структура витрат:

- витрати на розроблення, підтримку та вдосконалення програмного забезпечення для аналізування структури веб-застосунків на основі React.js;
- витрати на орендування серверів для програмного засобу;

– витрати на оплату праці;
– витрати на оренду офісного приміщення, опалення, освітлення, водопостачання;
– адміністративні витрати;
– витрати на рекламу та дослідження ринку.

Методом пошуку клієнтів було обрано використання тематичних ресурсів та спільнот, профільних видань, прямих контактів для продажів.

5.3. Розроблення маркетингової програми стартап-проекту

Маркетингова стратегія - елемент стратегії діяльності підприємства, спрямований на розробку, виробництво і доведення до покупця товарів і послуг, найбільш відповідних його потребам.

Маркетингова стратегія являє собою програму досягнення найголовніший мети компанії - забезпечення прибутку від ринкової діяльності. Мета - це заява про те, що саме хоче отримати (стабільно отримувати) компанія в доступній для огляду перспективі.

Маркетингова стратегія - комплекс принципів, за допомогою яких підприємство формує цілі маркетингу і організовує реалізацію цих цілей на ринку [43].

Маркетингова стратегія визначає темпи досягнення мети:

– швидке зростання прибутку від ринкової діяльності;
– стабільність зростання;
– скорочення прибутку з огляду на зростання деяких не ринкових показників, наприклад, капіталізації, глобалізації, присутності та ін.).

Найважливішим показником ринкової діяльності компанії є обсяг продажів, співвідношення між обсягом продажів і прибутковістю. Залежно

від того, як змінюється це співвідношення, маркетингова стратегія може передбачати:

- швидке зростання маркетингових показників;
- стабільну ринкову присутність;
- скорочення ринкової присутності.

Для досягнення стратегічно важливого для компанії обсягу продажів і прибутковості, необхідно поставити ряд маркетингових під цілей і визначити стратегію маркетингу щодо:

- збільшення частки ринку;
- скорочення частки ринку;
- необхідність присутності компанії в тому чи іншому ринковому сегменті;
- кількості покупців, їх типологію, ту чи іншу персоналізацію ставлення до певних клієнтів;
- впізнаваність, запам'ятовуваність і лояльність до марки, товару і товарного пропозицією [44].

Варіант з доступом для одноразової перевірки веб-застосунку актуальний для людей, що виступають його замовником, з метою фінальної перевірки перед підписанням акту виконаних робіт.

Варіант з місячною підпискою є актуальним для компаній, що створюють веб-застосунки, для перевірки веб-застосунків перед його демонстрацією клієнту.

Різниця в обмеженій кількості перевірок та необмеженій була створена для компаній, що мають певний цикл розробки програмного забезпечення.

Розрахований дохід по місяцям протягом першого року наведено в табл. 5.2. Всі суми наведено в гривнях.

Таблиця 5.2.

Доходи

	Одноразове використання програмного забезпечення	Місячна підписка з обмеженою кількістю перевірок	Місячна підписка з необмеженою кількістю перевірок	Всього
	1000	5000	10000	16000
2	3000	15000	20000	38000
3	5000	30000	60000	95000
4	10000	50000	100000	160000
5	10000	50000	200000	260000
6	12000	60000	400000	472000
7	12000	60000	400000	472000
8	12000	60000	400000	472000
9	15000	100000	700000	815000
10	15000	100000	700000	815000
11	25000	100000	700000	825000
12	27000	200000	700000	927000
				6850000

Сукупність загальних витрат складають наступні витрати:

- витрати на розроблення, підтримку та вдосконалення програмних продуктів (утримання робочих місць, придбання необхідних інструментів та програмних засобів);
- витрати на орендування серверів для програмного засобу;
- витрати на оплату праці;

– витрати на оренду офісного приміщення, опалення, освітлення, водопостачання;
– адміністративні витрати;
– витрати на рекламу та дослідження ринку.

Витрати по місяцям протягом першого року наведено в табл. 5.3. Всі суми наведено в гривнях.

Таблиця 5.3

Витрати

	Оренда обладнання	Оплата праці	Комунальні та адміністративні	Реклама	Всього
1	5000	35000	18000	10000	68000
2	5000	35000	18000	10000	68000
3	5000	35000	18000	10000	68000
4	5000	35000	18000	5000	63000
5	5000	35000	18000	5000	63000
6	5000	35000	18000	5000	63000
7	5000	15000	18000	3000	51000
8	5000	15000	18000	3000	51000
9	5000	15000	18000	3000	51000
10	5000	10000	18000	1000	34000
11	5000	10000	18000	1000	34000
12	5000	10000	18000	1000	34000
Сумарно за рік:					6250000

Розрахуємо маржинальний прибуток за перший рік за формулою (5.1).

$$\text{Маржинальний прибуток} = \text{Дохід} - \text{Витрати} \qquad (5.1)$$

За формулою (5.1) маржинальний прибуток за перший рік складе 7650000 - 6250000 = 600000 гривень.

5.4. Розроблення ринкової стратегії стартап-проекту

Ринкова стратегія - це план досягнення цілей підприємства, а також точка перетину сильних сторін компанії та можливостей, які надає ринок. Ринкова стратегія - це осередок маркетингу.

Ринкова стратегія включає в себе дві головні складові:

1. опис групи споживачів (сегмента), яким компанія може запропонувати очевидна перевага;
2. образ позиціонування пропозицій компанії, який повинен скластися у споживачів.

При всьому різноманітті стратегій Майкл Портер згрупував їх в три класи: стратегії лідируючих позицій по витратах, диференціації та концентрації.

1. Стратегія лідируючих позицій по витратах: компанія прагне до мінімальних прийнятним витрат на виробництво і розподіл продукції, щоб встановити більш низькі (в порівнянні з конкурентами) ціни і розширити свою частку ринку. Компанії, такі цій стратегії, повинні приділяти основну увагу розробці нової продукції, закупівель комплектуючих, виробництва та розподілення. Маркетингові навички необхідні їм в меншій мірі.
2. Стратегія диференціації: компанія прагне до досягнення переваги над конкурентами в істотно важливих для споживача областях. Компанія намагається завоювати лідируючі позиції в рівні послуг, в якості продукції, її оформленні або в технологіях.

3. Стратегія концентрації: увага компанії фокусується на одному або декількох вузьких сегментах ринку. Компанія прекрасно обізнана про потреби споживачів і в своїй діяльності дотримується стратегії лідируючих позицій по витратах або стратегії диференціації.

Для даного стартап проекту було обрано стратегію концентрації, оскільки програмне забезпечення виконує аналізування структури веб-застосунків, написаних тільки на основі React.js.

5.5. Висновки

В цьому розділі було описано ідею стартап проекту, а саме створення програмного забезпечення для аналізування структури веб-застосунку на основі React.js, через неспроможність розробників завжди дотримуватись корпоративних правил.

Були визначені зацікавлені сторони та такі ціннісні пропозиції як:

- доступ для одноразової перевірки веб-застосунку;
- місячна підписка з обмеженлю кількістю перевірок;
- місячна підписка з необмеженою кількістю перевірок.

ВИСНОВКИ

У магістерській дисертації розроблено програмне забезпечення для аналізування структури веб-застосунків на основі React.js. При цьому отримано такі результати:

1. Проаналізовано процес аналізування веб-застосунків. В результаті такого аналізу підтверджено необхідність створення відповідного програмного забезпечення. На основі результатів аналізу сформовано функціональні вимоги та поставлено цілі, які необхідно досягнути в процесі розроблення програмного забезпечення.

2. Побудовано концептуальну модель програмного забезпечення аналізування структури веб-застосунків на основі React.js, використання якої дозволяє формалізувати його роботу на рівні функцій, процесів і потоків даних, а також сформувати та обґрунтувати варіанти використання програмного забезпечення.

3. Побудовано фізичну модель програмного забезпечення на основі формалізування його роботи на рівні функцій, процесів і потоків даних, використання якої дозволяє надати нову якість програмному забезпеченню при створенні або вдосконаленні, зокрема, забезпечити його функціональну придатність до аналізування структури веб-застосунків на основі React.js.

4. Наведено контрольні приклади для проведення тестування програмного забезпечення. Для цього проведено функціональне тестування шляхом обирання та описання відповідних характеристик та метрик якості програмного забезпечення.

5. Визначено ринкові перспективи проекту, графік та принципи організації виробництва, фінансовий аналіз. Узагальнено етапи розроблення стартап-проекту для розроблення та виведення стартап-проекту на ринок передбачає здійснення низки кроків.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Веб-застосунок [Електронний ресурс]. – Режим доступу: https://goo.gl/8oxyMn.
2. React.js [Електронний ресурс]. – Режим доступу: https://goo.gl/fs11mV.
3. Статичний аналіз коду [Електронний ресурс]. – Режим доступу: https://www.techopedia.com/definition/24621/static-code-analysis.
4. Динамічний аналіз коду [Електронний ресурс]. – Режим доступу: https://www.techopedia.com/definition/30015/dynamic-code-analysis.
5. Integrate static analysis into a software development process / Walter W. Schilling, Jr. and Mansoor Alam [Електронний ресурс]. – Режим доступу: https://www.embedded.com/design/prototyping-and-development/4006735/Integrate -static-analysis-into-a-software-development-process.
6. ESLint [Електронний ресурс]. – Режим доступу: https://eslint.org/docs/about.
7. JSHint [Електронний ресурс]. – Режим доступу: https://jshint.com/about/.
8. Рябченко Д.О. Програмний засіб аналізування структури веб-застосунків на основі React.js / Д.О. Рябченко, В.В. Цуркан // XI наукова конференція магістрантів та аспірантів «Прикладна математика та комп’ютинг» (ПМК2018-2), 2018 – 238 с.
9. IDEF0 [Електронний ресурс]. – Режим доступу: https://goo.gl/m2Bzk3.
10. IDEF3 [Електронний ресурс]. – Режим доступу: https://goo.gl/u38Qnz.

11. DFD [Електронний ресурс]. – Режим доступу: https://www.lucidchart.com/pages/data-flow-diagram.

12. McLaughlin B. Head First Object Oriented Analysis and Design / McLaughlin, B., Pollice, G., West, D. McLaughlin, B., Pollice, G., West, D. - O'Reilly Media, Inc., 2006. – 297 p.

13. Siau, K. Are use case and class diagrams complementary in requirements analysis? An experimental study on use case and class diagrams in UML / K. Siau, L. Lee // Requirements Engineering. – 2004. – № 9(4). – 234 p.

14. Use cases diagram [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Use_case_diagram.

15. Логічна структура [Електронний ресурс]. – Режим доступу: https://goo.gl/A5iWzj.

16. Логічна структура [Електронний ресурс]. – Режим доступу: http://www.maksakov-sa.ru/DiplProekty/SoderganDiplProeta/LogichStruct/index.html.

17. Javascript [Електронний ресурс]. – Режим доступу: https://www.javascript.com/.

18. Opportunities of Javascript [Електронний ресурс]. – Режим доступу: https://htmlacademy.ru/courses/javascript.

19. AJAX [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Ajax_(programming).

20. Javascript [Електронний ресурс]. – Режим доступу: https://learn.javascript.ru/intro.

21. React [Електронний ресурс]. – Режим доступу: https://reactjs.org/tutorial/tutorial.html.

22. JSX [Електронний ресурс]. – Режим доступу: https://learn-reactjs.ru/basics/introduction-to-jsx

23. Clark A. React v16.0§New render return types: fragments and strings [Електронний ресурс]. – Режим доступу: https://reactjs.org/blog/2017/09/26/react-v16.0.html#new-render-return-types-fragments-and-strings.

24. Clark A. React v16.0§Support for custom DOM attributes [Електронний ресурс]. – Режим доступу: https://reactjs.org/blog/2017/09/26/react-v16.0.html#support-for-custom-dom-attributes.

25. Benefits of JSX in React [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/React_(JavaScript_library)#JSX.

26. Kaner C. Exploratory Testing (PDF). Quality Assurance Institute Worldwide Annual Software Testing Conference [Електронний ресурс]. – Режим доступу: http://www.kaner.com/pdfs/ETatQAI.pdf.

27. Software testing [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Software_testing.

28. Thick Client Definition [Електронний ресурс]. – Режим доступу: http://www.techterms.com/definition/thickclient.

29. Товстий клієнт [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Fat_client.

30. Node.js [Електронний ресурс]. – Режим доступу: https://nodejs.org/en/.

31. WebStorm [Електронний ресурс]. – Режим доступу: https://www.jetbrains.com/webstorm/.

32. Git [Електронний ресурс]. – Режим доступу: https://git-scm.com/.

33. ASP.NET [Електронний ресурс]. – Режим доступу: https://www.asp.net/.

34. ColdFusion [Електронний ресурс]. – Режим доступу: https://helpx.adobe.com/support/coldfusion.html.

35. PHP [Електронний ресурс]. – Режим доступу: http://www.php.net/.

36. JSP [Електронний ресурс]. – Режим доступу: https://www.oracle.com/technetwork/java/index-jsp-138231.html.

37. Notepad [Електронний ресурс]. – Режим доступу: https://notepad-plus-plus.org/.

38. Acrobat Reader [Електронний ресурс]. – Режим доступу: https://get.adobe.com/ru/reader/otherversions/.

39. What is web applications? [Електронний ресурс]. – Режим доступу: https://www.quora.com/What-is-web-application.

40. Web applications [Електронний ресурс]. – Режим доступу: https://www.maxcdn.com/one/visual-glossary/web-application/.

41. Hamza M. Advantages of Developing Modern Web apps with React.js [Електронний ресурс]. – Режим доступу: https://medium.com/@hamzamahmood/advantages-of-developing-modern-web-apps-with-react-js-8504c571db71.

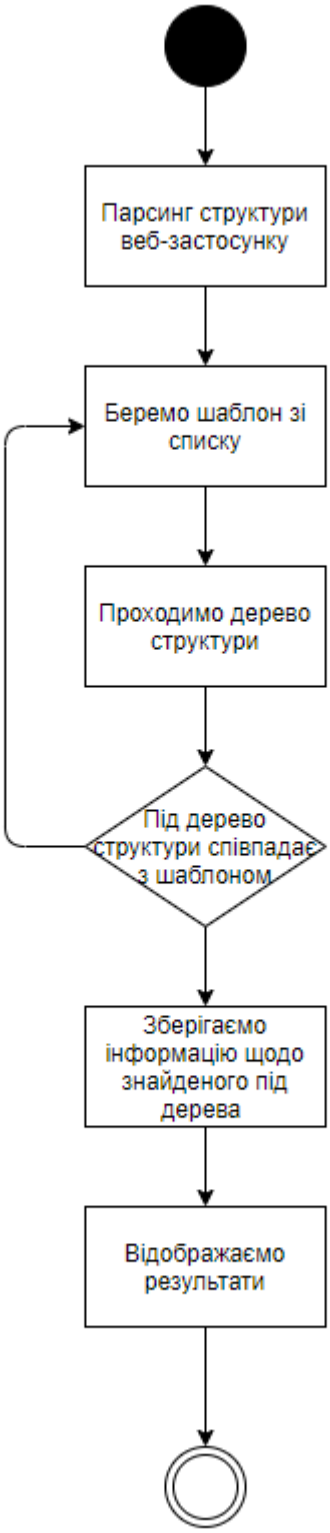
42. Зацікавлені сторони [Електронний ресурс]. – Режим доступу: http://pmexperience.org/ru/content/glava-2-standart-ansi-pmi-pmbok-zainteresovannye-storony-i-osnovnye-deystvuyushchie-lica-proekta.

43. Практичне використання маркетингової стратегії [Електронний ресурс]. – Режим доступу: https://kakzarabativat.ru/marketing/marketingovaya-strategiya/.

44. Маркетингова стратегія [Електронний ресурс]. – Режим доступу: http://www.marketch.ru/marketing_dictionary/marketing_terms_m/abc_marketing_strategy/.

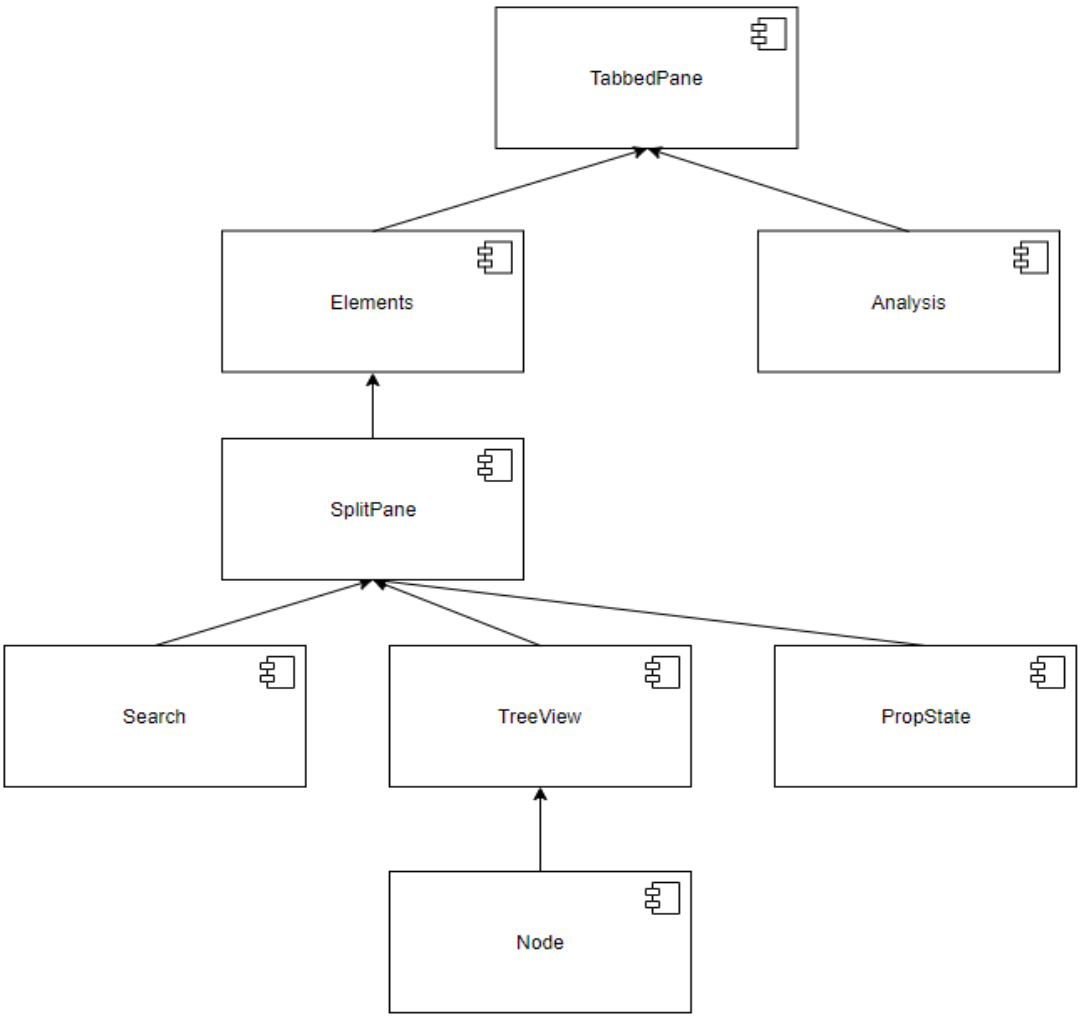
ДОДАТОК 1

Графічні матеріали



Діаграма діяльності

Рябченко Д.О., КП-71мп



Діаграма компонентів

Рябченко Д.О., КП-71мп

ДОДАТОК 2

Фрагмент тексту програми

TabbedPane.js

```
const PropTypes = require('prop-types');
const React = require('react');
const decorate = require('./decorate');
const {sansSerif} = require('./Themes/Fonts');

import type {Theme} from './types';

type Props = {
  tabs: {[key: string]: () => React.Node},
  selected: string,
  setSelectedTab: (name: string) => void,
};

class TabbedPane extends React.Component<Props> {
  context: {
    theme: Theme,
  };

  render() {
    var {theme} = this.context;
    var tabs = Object.keys(this.props.tabs);
    if (tabs.length === 1) {
```

```
    return this.props.tabs[tabs[0]]();
  }
  return (
    <div style={styles.container}>
      <ul style={tabsStyle(theme)}>
        {tabs.map((name, i) => (
          <li
            key={name + i}
            onClick={() => this.props.setSelectedTab(name)}
            style={tabStyle(name === this.props.selected, theme)}
          >
            {name}
          </li>
        ))}
      </ul>
      <div style={styles.body}>
        {this.props.tabs[this.props.selected]()}
      </div>
    </div>
  );
 }
}

TabbedPane.contextTypes = {
 theme: PropTypes.object.isRequired,
};

const tabsStyle = (theme: Theme) => ({
```

```
  display: 'flex',
  flexShrink: 0,
  listStyle: 'none',
  margin: 0,
  backgroundColor: theme.base01,
  borderBottom: `1px solid ${theme.base03}`,
  padding: '0 0.25rem',
});

const tabStyle = (isSelected: boolean, theme: Theme) => {
  return {
    padding: '0.25rem 0.75rem',
    lineHeight: '15px',
    fontSize: sansSerif.sizes.normal,
    fontFamily: sansSerif.family,
    cursor: 'pointer',
    borderTop: '1px solid transparent',
    borderBottom: isSelected ? `2px solid ${theme.state00}` : 'none',
    marginBottom: isSelected ? '-1px' : '1px',
  };
};

var styles = {
  container:{
    flex: 1,
    display: 'flex',
    flexDirection: 'column',
    width: '100%',
```

```
  },
  body: {
    flex: 1,
    display: 'flex',
    minHeight: 0,
  },
};

module.exports = decorate({
  listeners: () => ['selectedTab'],
  shouldUpdate: (props, prevProps) => {
    for (var name in props) {
      if (props[name] !== prevProps[name]) {
        return true;
      }
    }
    return false;
  },
  props(store) {
    return {
      selected: store.selectedTab,
      setSelectedTab: name => store.setSelectedTab(name),
    };
  },
}, TabbedPane);
```

Analysis.js

```
const React = require('react');
```

```
const {List} = require('immutable');
const decorate = require('./decorate');
const pattern1 = require('../patterns/pattern1');
const pattern2 = require('../patterns/pattern2');

const my_tree = {};

const buildTree = function(store, root, id) {
  const node = store.get(id);
  const children = node.get('children');
  root = {
    name: node.get('name'),
    props: node.get('props'),
    state: node.get('state'),
    id,
  };
  if (children && children.map) {
    root.children = children.map(_id => buildTree(store, {}, _id));
  } else if (typeof children === 'string') {
    root.children = [{
      name: children,
    }];
  }

  return root;
};

const patterns = [pattern1, pattern2];
```

```
let recList = [];

const analysMain = function(treeRoot) {
  recList = [];
  for (const p of patterns) {
    analysPath(treeRoot, p);
  }
};

const analysPath = function(node, pattern) {
  const {name, description, structure} = pattern;
  if (structure.name && structure.name === 'any' &&
    node.children &&
    node.name !== structure.not) {
    for (const c of node.children) {
      if (analysCheck(c, structure.child)) {
        recList.push({
          name, description, node: `${node.name} -> ${c.name}`, id: node.id,
        });
      }
    }
  }
  if (structure.state === true &&
    node.state && node.children) {
    for (const c of node.children) {
      if (analysCheck(c, {...structure.child, oneOf: Object.keys(node.state)})) {
        recList.push({
          name, description, node: `${node.name} -> ${c.name}`, id: node.id,
```

```
        });
      }
    }
  }
  if (node.children) {
    for (const c of node.children) {
      analysPath(c, pattern);
    }
  }
};

const analysCheck = function(node, pattern) {
  if (pattern.name && pattern.name === node.name) {
    if (pattern.child && node.children) {
      return node.children.reduce((acc, cur) => {
        if (analysCheck(cur, pattern.child)) {
          return true;
        }
        return acc;
      }, false);
    }
    return true;
  }
  if (pattern.props === true && node.props && pattern.oneOf) {
    for (const p of Object.keys(node.props)) {
      if (pattern.oneOf.includes(p)) {
        if (pattern.child && node.children) {
          return node.children.reduce((acc, cur) => {
```

```
          if (analysCheck(cur, {...pattern.child, oneOf: pattern.oneOf})) {
            return true;
          }
          return acc;
        }, false);
      } else if (!node.children) {
        return false;
      }
      return true;
    }
  }
  return false;
};

class Analysis extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      results: [],
    };
  }

  componentDidMount() {
    this.runScript();
  }

  runScript() {
```

```
    const {roots, store} = this.props;
    roots.forEach(id => {
      analysMain(buildTree(store, my_tree, id));
    });
    this.setState({results: recList});
  }

  render() {
    return (
      <div>
        <h4>Results of the analysis:</h4>
        {this.state.results ? this.state.results.map((rec, i) => (<div key={i}
      onClick={() => {
          this.props.store.searchRoots = List([rec.id]);
          this.props.store.setSelectedTab('Elements');
        }} style={{
          borderBottom: '1px solid rgb(218, 218, 218)',
          borderTop: '1px solid rgb(218, 218, 218)',
          cursor: 'pointer'}}>{rec.node}: {rec.description}</div>)) : 'Patterns didn\'t
      find any matches'}
      </div>
    );
  }
}

module.exports = decorate({
  listeners() {
    return ['roots'];
```

```
  },
  props(store) {
    return {
      roots: store.roots,
      store,
    };
  },
}, Analysis);
```

TreeView.js

```
const Node = require('./Node');
const PropTypes = require('prop-types');
const React = require('react');
const SearchUtils = require('./SearchUtils');
const Breadcrumb = require('./Breadcrumb');

const decorate = require('./decorate');
const {monospace, sansSerif} = require('./Themes/Fonts');

import type {List} from 'immutable';
import type {Theme} from './types';

const MAX_SEARCH_ROOTS = 200;

type Props = {
  reload?: () => void,
  roots: List,
```

```
  searching: boolean,
  searchText: string,
};

class TreeView extends React.Component<Props> {
  node: ?HTMLElement;

  getChildContext() {
    return {
      scrollTo: this.scrollTo.bind(this),
    };
  }

  scrollTo(toNode) {
    if (!this.node) {
      return;
    }
    let val = 0;
    const height = toNode.offsetHeight;
    while (toNode && this.node.contains(toNode)) {
      val += toNode.offsetTop;
      toNode = toNode.offsetParent;
    }
    const top = this.node.scrollTop;
    const rel = val - this.node.offsetTop;
    const margin = 40;
    if (top > rel - margin) {
      this.node.scrollTop = rel - margin;
```

```
  } else if (top + this.node.offsetHeight < rel + height + margin) {
    this.node.scrollTop = rel - this.node.offsetHeight + height + margin;
  }
}

render() {
  const {theme} = this.context;

  if (!this.props.roots.count()) {
    if (this.props.searching) {
      return (
        <div style={styles.container}>
          <span style={noSearchResultsStyle(theme)}>No search results</span>
        </div>
      );
    } else {
      return (
        <div style={styles.container}>
          <div ref={n => this.node = n} style={styles.scroll}>
            <div style={styles.scrollContents}>
            Waiting for roots to load...
            {this.props.reload &&
              <span>
                to reload the inspector <button onClick={this.props.reload}> click
    here</button>
              </span>}
            </div>
          </div>
```

```
      </div>
    );
  }
}

// Convert search text into a case-insensitive regex for match-highlighting.
const searchText = this.props.searchText;
const searchRegExp = SearchUtils.isValidRegex(searchText)
  ? SearchUtils.searchTextToRegExp(searchText)
  : null;

if (this.props.searching && this.props.roots.count() >
  MAX_SEARCH_ROOTS) {
  return (
    <div style={styles.container}>
      <div ref={n => this.node = n} style={styles.scroll}>
        <div style={styles.scrollContents}>
          {this.props.roots.slice(0, MAX_SEARCH_ROOTS).map(id => (
            <Node
              depth={0}
              id={id}
              key={id}
              searchRegExp={searchRegExp}
            />
          )).toJS()}
          <span>Some results not shown. Narrow your search criteria to find
  them</span>
        </div>
```

```
        </div>
      </div>
    );
  }

  return (
    <div style={styles.container}>
      <div ref={n => this.node = n} style={styles.scroll}>
        <div style={styles.scrollContents}>
          {this.props.roots.map(id => (
            <Node
              depth={0}
              id={id}
              rootId={id}
              key={id}
              searchRegExp={searchRegExp}
            />
          )).toJS()}
        </div>
      </div>
      <Breadcrumb />
    </div>
  );
 }
}

TreeView.childContextTypes = {
 scrollTo: PropTypes.func,
```

```
};

TreeView.contextTypes = {
  theme: PropTypes.object.isRequired,
};

const noSearchResultsStyle = (theme: Theme) => ({
  color: theme.base04,
  fontFamily: sansSerif.family,
  fontSize: sansSerif.sizes.large,
  fontStyle: 'italic',
  padding: '0.5rem',
});

const styles = {
  container: {
    fontFamily: monospace.family,
    fontSize: monospace.sizes.normal,
    lineHeight: 1.5,
    flex: 1,
    display: 'flex',
    flexDirection: 'column',
    minHeight: 0,

    WebkitUserSelect: 'none',
    MozUserSelect: 'none',
    userSelect: 'none',
  },
```

```
  scroll: {
    overflow: 'auto',
    minHeight: 0,
    flex: 1,
    display: 'flex',
    flexDirection: 'column',
    alignItems: 'flex-start',
    padding: '0.5rem 0.25rem',
  },

  scrollContents: {
    flexDirection: 'column',
    flex: 1,
    display: 'flex',
    alignItems: 'stretch',
    width: '100%',
  },
};

const WrappedTreeView = decorate({
  listeners(props) {
    return ['searchRoots', 'roots'];
  },
  props(store, props) {
    return {
      roots: store.searchRoots || store.roots,
      searching: !!store.searchRoots,
```

```
      searchText: store.searchText,
    };
  },
}, TreeView);

module.exports = WrappedTreeView;
```